REPORT DOCUMENTATION PAGE

38

AFRL-SR-BL-TR-98-

0862

Public reporting burden for this collection of information is estimated to average 1 hour per respon
gathering and maintaining the data needed, and completing and reviewing the collection of informa
collection of information, including suggestions for reducing this burden, to Washington Headquart
Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Buc

data sources,
aspect of this
1215 Jefferson
20503.

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE

3. REPORT TYPE AND DATES COVERED
20 Sep 96 to 19 Sep 98 (Final)

4. TITLE AND SUBTITLE
(SBIR 95-11) Circuits and Devices for High-Speed instrumentation

5. FUNDING NUMBERS
65502F
3005/SS

6. AUTHOR(S)
Dr Marsland

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Focused Research Inc
2630 Walsh Avenue
Santa Calra Ca 95051-0905

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NE
801 North Randolph Street Rm 732
Arlington, VA 22203-1977

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
F49620-96-C-0052

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVAL FOR PUBLIC RELEASED; DISTRIBUTION UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

We have developed GaAs-based resonant tunneling diodes (RTDs) based on second quantum well level tunneling that achieve 200kA/cm 2 with 1.2 V peark voltage. We have shown that these results are reproducible by exploring a parameter space of quantum-well widths and barrier heights for this structure. A picosecond sampling oscilloscope was also developed. With a low-cost high-stability time-base and a 100 GHz sampling aperture created by the nonlinear transmission line driver, we have created a test vehicle that can be used for evaluating furture RTD-circuit developments.

14. SUBJECT TERMS

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

FINAL REPORT

Circuits and Devices for High-Speed Instrumentation

December 28, 1998

Sponsored by

Air Force Office of Scientific Research (AFOSR)

Under Contract:

F49620-96-C-0052

Principal Investigator:

Robert A. Marsland
Focused Research, Inc.
555 Science Dr.
Madison, WI 53711
608-238-2455

Authors: Grant Emmel, Robert Marsland, and Ali Mirabedini

Contract sequence number for this data item: 0002AA

"The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government."

19990115 094

Abstract

We have developed GaAs-based resonant tunneling diodes (RTDs) based on second quantum well level tunneling that achieve >200 kA/ cm 2 with < 1.2 V peak voltage. We have shown that these results are reproducible by exploring a parameter space of quantum-well widths and barrier heights for this structure. A picosecond sampling oscilloscope was also developed. With a low-cost high-stability time-base and a >100 GHz sampling aperture created by the nonlinear transmission line driver, we have created a test vehicle that can be used for evaluating future RTD-circuit developments.

DTIC QUALITY INSPECTED 3

Contents

DTIC QUALITY INSPECTED 3

Figures and Tables

Preface and Acknowledgments

As with any research program of this scope, it requires the dedication and skill of a large number of people. The remarkable progress on RTD device development was due to the attention of Professors Dan Botez and Luke Mawst of the UW-Madison and graduate student Ali Mirabedini. Robert Marsland and Grant Emmel handled the NLTL-sampler and time-base development with some timely assistance from Andrew Davidson and Robert S. Williamson III. Packaging and wire-bonding was professionally attended by Terri Ripley and Pauline Prather. We did not get very far with the final integration of RTD and Schottky because of the unexpected difficulties with the planar RTD, but this is proceeding under separate funding.

The authors would like to thank the following for their key contributions:

Dr. Howard Schlossberg of the Air-Force Office of Scientific Research – his patience and encouragement toward commercially useful results have been an inspiration throughout the program.

Dr. Milton Chang – his trust allowed us to spend two years running down this path of ultra-fast electronics. Few, if any, commercial corporate managers are so generous.

Prof. Mark Rodwell of UCSB – his assistance in constructing the work plan and with rf-probe station development gave us a quick start.

Mr. Andrew Cordes -- his support of the New Focus firmware design gave us an excellent position for launching a product based on this research.

Summary

This two-year program was composed of three major pieces: resonant-tunneling diode (RTD) development, nonlinear transmission line (NLTL)- based test-system development, and final integration of RTDs and Schottky-diode sampling circuits. The following major accomplishments:

RTD Development

- MOCVD-grown, GaAs based RTD with a achieves 1.2V peak voltage and greater than 200 kA/cm^2 peak current density in a non-planar but reproducible and well-documented process. Work on the planar process is continuing under separate funding.

NLTL-based Test-System Development

- NLTL-based sampling system with low-cost ECL delay generator achieves sub-picosecond stability.
- NLTL-based system measures detector and package-limited 8-ps pulse compared to 12 ps with best commercial equipment.
- NLTL-based test system development continues under separate follow-on funding.

Final Integration

- Monolithically integrated RTD driven Schottky-diode sampling gate with a goal of 6 ps aperture and 200 mV dynamic range consuming less than 0.05 mm^2 excluding bond-pads have been designed and are in fabrication. Additional funding may permit evaluation of these samples.

Introduction

Presently, 1-Gbit/second fiber-optic local-area networks are beginning to gain widespread acceptance and DARPA is pushing for 160 Gbit/second future generation technology. GaAs HBT logic chips are becoming available that clock at 34 GHz. Research into parallel optical networks goes into high-gear with the first results of the POLO program becoming commercially available. The National Ignition Facility proposed by LLNL requires 10 Gspl/sec transient digitizers for power balancing in its 192 beams. Meanwhile, scientists working with ultra-fast phenomena continue to use the streak camera as the primary measurement device despite it's non-linear time-base, limited dynamic range, high-cost, and poor performance at longer wavelengths.

The common thread in all the above developments is the need for a calibrated receiver capable of measuring optical or electrical waveform information with picosecond resolution at a reasonable cost. Although an electrical sampling circuit has been demonstrated with a bandwidth of 700 GHz [1] in the laboratory, commercial sampling oscilloscopes are limited to ~50 GHz by the combination of trigger stability, speed and accuracy of the sampling gate, and the front-panel connector bandwidth. These units sell for over $30,000. For optical signals, the maximum bandwidth is limited to ~40 GHz by the photodetector/ scope combination. At the same time, commercially available Nyquist sampling rate digital oscilloscopes are limited to 2 GHz analog bandwidth (8 Gspl/sec) and sell for $9,950 a channel in a $45,900 mainframe.

Clearly a faster oscilloscope with an input photodiode of comparable speed is necessary to meet the needs of the ultrafast TDM and laser physics research. For parallel optical networks involving 12 to 32 channels that need to be analyzed simultaneously, it is the price-tag and lack of multiple optical inputs that makes present instrumentation unworkable. The results of this program will address both of these problems and pave the way for a faster Nyquist sampling rate oscilloscope by developing the enabling integrated circuits and systems shown in Table 1.

[1] M.J.W. Rodwell, S. T. Allen, R. Y. Yu, M. G. Case, M. Reddy, E. Carman , J. Pusl, M. Kamegawa, Y. Konishi, and R. Pullela, "Active and Nonlinear Wave Propagation Devices in Ultrafast Electronics and Optoelectronics", *Proceedings of the IEEE*,

Table 1: Possible end-products and their enabling technologies

Product	Enabling Technologies	⇒Advantage
>100 GHz sampling oscilloscope w/ optical input and > 100GHz triggering ability **(3x state of art for optical input bandwidth)**	a) RTD Sample/ Hold, equivalent time operation b) NLTL Sample /Hold, equivalent time operation c) Equivalent-time time-base	⇒ 3ps, estimated IC cost $5 each ⇒ 0.5ps estimated IC cost $40 each, + requires power amp ⇒ 0.1ps stability, estimated cost $1000 for complete board including micro-controller
> 40 GHz 12-channel Oscilloscope array w/ optical input **(12x state of art for # of optical inputs)**	a) Arrayed RTD Sample/ Hold, equivalent time operation b) RTD shift register for sequential gate strobe	⇒ 10ps, estimated cost $20 each for array of 12 ⇒ on-chip logic reduces packaging cost. No speed requirement.
> 15 GHz Transient digitizer **(8x state of art for analog signal bandwidth)**	a) Arrayed RTD Sample/ Hold, single-shot operation b) RTD shift register for sequential gate strobe	⇒ 10ps, estimated cost $100 each for array of 32 ⇒ 10 ps delay per shift, integrated with samplers

The technologies and uses listed Table 1 are determined by the IC device technology as shown in Table 2. Although the NLTL has the fastest rise-time and slew-rate, the propagation delay is long leading to large recovery time and circuit area. This makes the NLTL the choice for sampling apertures <1ps at steady state sampling rates and low levels of integration. When the switching time needs to vary at speeds greater than 1 GHz, or integration is required, the RTD or HBT become necessary. The HBT is preferred, and necessary when unilateral gain is required but far more difficult to fabricate and integrate. The RTD is the clear winner when cost and time-to-market are considerations. The RTD also holds promise for very high levels of integration where its low valley current provides low "stand-by" power dissipation [2]

[2] C.E. Chang, P.M. Asbeck, K.-C. Wang, E.R. Brown, "Analysis of Heterojunction Bipolar Transistor/ Resonant Tunneling Diode Logic for Low-Power and High-Speed Digital Applications," *IEEE Transactions on Electron Devices*, Vol 40, No. 4, pp. 685-691 (1993).

Table 2: Comparison of various high-speed devices and circuits.

Device/ Circuit	Area (μm^2)	Rise-time (ps)	Voltage swing (V)	Recover time (ps)	Slew-rate (V/ps)
NLTL	4x10^6	0.5	3	1000	6
RTD	36	3	1	3	0.3
HBT	6	3	2	3	0.6
X-former	2x10^3	0.5	-	100	-
Sampler	1x10^3	0.5	-	10	-

Methods, Assumptions, and Procedures

RTD Development

Objective: MOCVD-grown, GaAs based RTD with a goal of greater than 1V peak voltage and greater than 200 kA/cm^2 peak current density in a quasi-planar, manufacturable, process.

Recent research has led to exploring materials other than GaAs/AlGaAs to improve device parameters such as voltage swing and peak current density and peak-to-valley ratio [3, 4, 5, 6, 7]. Although the early work in RTD material growth was performed exclusively by MBE, researchers are beginning to turn toward MOCVD as a possible source of high-quality lower-cost material [8]. MOCVD can provide excellent uniformity and is a lower cost production process than MBE. During the Phase I contract, MOCVD was used to grow a unique strained InGaAs RTD structure that succeeded where others have failed due to the excellent AlGaAs/ InGaAs interfaces.

Figure 1 (a) shows the TEM for a AlGaAs/InGaAs/GaAs structure grown with 16Å-thick barriers and 57Å-thick well. The high resolution lattice-matched TEM image, Figure 1 (b), estimates the thickness of the first barrier (from bottom), well, and top barrier as 14.5 Å, 57 Å, and 13 Å, respectively, which are in good agreement with the target values.

This same MOCVD approach was used in the Phase II to further refine the RTD performance. The following improvements were required:

1) Device peak voltage should be closer to 1V or less;

2) Device material growth should be reproducible;

3) Device breakdown voltage should provide safe operation in a 50 ohm system;

4) The device should be compatible with a planar process.

The approach for items one and two was to grow a series of devices with well widths varying from 55 to 61 Angstroms and barrier widths from 21 to 30 Angstroms.

[3] Ozbay, et al. "1.7-ps, microwave, integrated-circuit-compatible InAs/AlSb resonant tunneling diodes." IEEE *Electron Device Letters* (Aug. 1993) vol.14, no.8, p. 400-2.

[4] Smith, et al. "0.1 μm Schottky-collector AlAs/GaAs resonant tunneling diodes." *IEEE Electron Device Letters*, vol. 15, no. 8, pp. 295-7 (1994).

[5] Soderstrom, et al. "Growth and characterization of high current density, high-speed InAs/AlSb resonant tunneling diodes." *Applied Physics Letters,* vol.58, no.3, pp. 275-7 (1991).

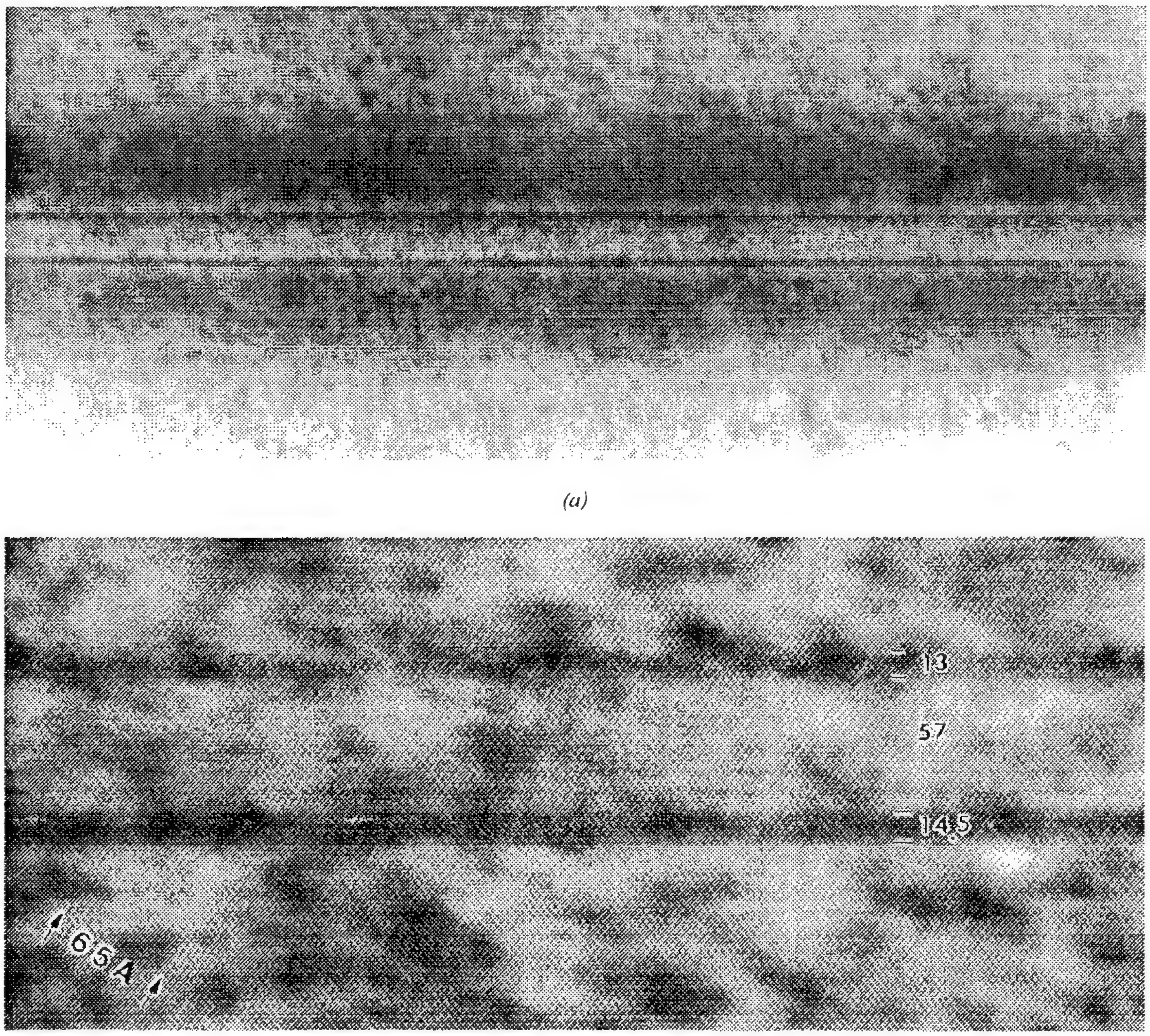
[6] Smet, T.P.E. Broekaert, and Clifton G. Fonstad, "Peak-to-valley current ratios as high as 50:1 at room temperature in pseudomorphic $In_{0.53}$ $Ga_{0.47}$ As / AlAs / InAs resonant tunneling diodes." *J. Appl. Phys.* vol 71, no. 5, pp. 2475-7 (1992).

[7] Broekaert and C.G. Fonstad, "$In_{0.53}$ $Ga_{0.47}$ As / AlAs resonant tunneling diodes with peak current densities in excess of 450 kA/ cm^2." *J. Appl. Phys.* vol. 68, no. 8, pp. 4310-2 (1990).

[8] Keller, J.C. Yen, S.P. DenBaars, and U.K. Mishra, "Ga_x In_{1-x} As/AlAs resonant tunneling diodes grown by atmospheric pressure metalorganic chemical vapor deposition." *Appl. Phys. Lett.* vol 65, no. 17, pp. 2159-2161 (1994).

Table 3: Matrix of experiments for RTD development

	Well Width (□)		
Barrier Width (□)	55	58	61
21			X
24		X	
30	X	X	X



(a)

(b)

Figure 1: Transmission-electron micrograph (TEM) (a) and high-resolution TEM (b) showing estimated layer ticknesses based on the lattice constant of GaAs.

The operating assumption was that thicker barriers would provide more reproducible results and less heating problems while thicker well width would reduce peak voltage by reducing the voltage necessary to line up the quantum well state to the emitter.

The problem addressed in item three is the very sharp increase in current after the valley of the I-V curve. The increase is so abrupt that safety considerations reduce the useful voltage swing of the device. The method here was to modify the I-V curve through doping of the RTD to allow operation over a wider range of voltages. Devices were grown with various doping profiles to simultaneously optimize low peak-voltage and high breakdown.

Finally, the entire RTD structure needed to be adapted to growth on a semi-insulating substrate to allow co-integration with other devices such as Schottky diodes. The approach was to design a mask-set that included a variety of RTD test structures as well as complete circuits designed to accommodate a wide range of RTD parameters.

NLTL-based Test-System Development

Objective: 3 ps optical sampling oscilloscope with sub-picosecond stability for use as a known test vehicle for the RTD driven circuits.

A sampling oscilloscope with ~2ps risetime is required for testing all of the RTD circuits to be developed. In addition, a sampling oscilloscope with 2 ps risetime and a photodiode front end would be an excellent product spin-off from this program. Sampling oscilloscopes consist of a sampling device, a triggered timebase, memory for sample storage, and a display Figure 2. Sampling oscilloscope bandwidth is determined by the sampling device and by the timebase.

The pulse generators are in the form of GaAs integrated circuits, containing a microwave transmission line loaded by Schottky diodes. Nonlinear electrical wave propagation on these devices, referred to as Nonlinear Transmission Lines, or NLTLs, results in the compression of input electrical pulses during propagation to the NLTL output. Pulses of several volts amplitude and a few picoseconds duration are readily formed. These pulses are used to momentarily gate a Schottky-diode sampling bridge fabricated on the same wafer. The combination is effectively the high-speed portions of a sampling oscilloscope on an IC.

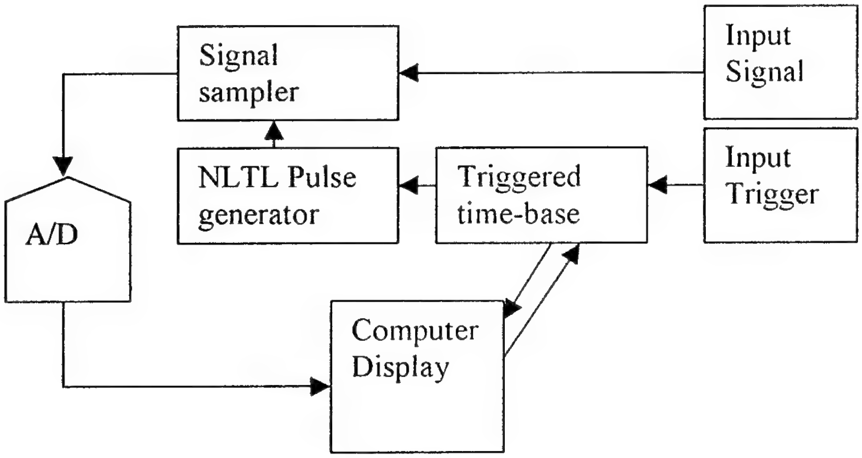


Figure 2: General block diagram of a sampling oscilloscope.

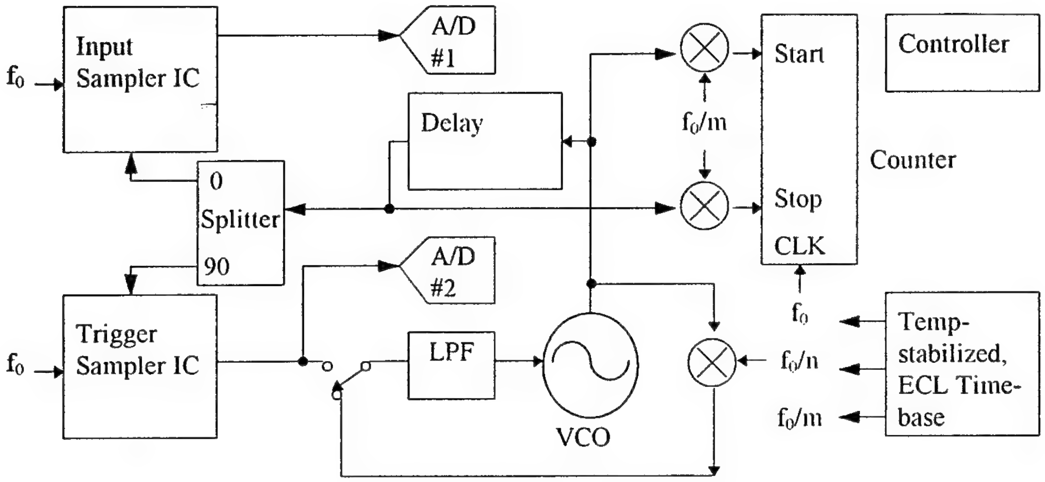


Figure 3: Low-cost, precision time base and delay architecture for demonstrating the sampling oscilloscope.

To fully demonstrate the trigger and sampling oscilloscope function as in Figure 2 it is necessary to have an accurate, sub-picosecond resolution time base. One method for achieving the trigger portion is based on a sampling phase detector. This technique uses an NLTL driven sampler to mix the input trigger signal with a comb of harmonics spaced at ~10 MHz. If the input signal lies somewhere in the range of 10MHz to 300 GHz, and the sampled output is fed back to the 10 MHz voltage-controlled oscillator, the system should lock onto some unknown harmonic. It isn't important which harmonic is locked on as long as it is a harmonic of the 10 MHz oscillator and not a harmonic of the trigger signal fundamental. This technique is expected to be very low cost and work well when the trigger signal is highly periodic.

For more general applications, the more traditional trigger recognizer will be required and will be implemented in the same RTD technology as the sampling gate driver. In either case, the timing offsets must be provided by a repeatable delay line. Figure 3 shows a block diagram of the approach to achieving the low-cost, sub-picosecond time base. The low cost comes from the use of high-volume ECL logic and PLL chips now available from Motorola. The stability comes from the use of New Focus temperature control technology developed for the tunable laser product line.

The range is 4 ns with a resolution of approximately 0.1 picosecond so 16 bits are needed to set the full range of adjustment. The Motorola Eclipse Logic MC100E196/195 Programmable Delay Line has the capability to achieve these goals. The 196 can be cascaded, hence two will be used to give the range of 4ns with 7 bits. The remaining 9 bits will be achieved using the MC100E196 fine-tune input. For a resolution of 0.1 ps (using 20 ps time range/0.6V voltage range from the Motorola Data Sheet) a voltage resolution of 3 mV is required which can readily be achieved with a 10 bit D/A converter using a 1V reference.

However, the delay times must be stable over temperature and voltage variations. According to Motorola documentation on the sensitivity of the 196 to temperature and voltage variations, one can derive an equation.

$[(4.7\text{ps/C})*\delta T] + [(3.5\text{ps/V})*\delta V] = \delta t$

where C is degrees Celsius, V is power supply volts. So, for δt better than 0.1ps, δT must be less than 0.01 C and δV must be less than 3 mV.

For voltage regulation, off the shelf components will give somewhere between 15mV and 30mV (ref. Burr Brown REG1117-5 and Analog Devices ADM663A). It is not unreasonable to believe that clever circuit design (such as some sort of voltage feedback from a stable reference) could achieve at least an order of magnitude improvement in regulation. For temperature regulation, the problem is more severe. Fortunately, this is a problem that has been addressed in another highly temperature sensitive product, the New Focus external-cavity diode laser. Here the laser diode element is routinely kept to within 0.01C to maintain laser frequency stability. We will apply these same techniques to the stabilization of the ECL delay parts to maintain fundamental stability, it is expected that a product based on this design would also incorporate a self-calibration feature to remove long-term drifts in temperature and bias.

Final Integration

Objectives:

- Monolithically integrated RTD driven Schottky-diode sampling gate with a goal of 6 ps aperture and 200 mV dynamic range consuming less than 0.05 mm^2 excluding bond-pads;
- Monolithically integrated RTD driven Schottky-diode sampling gate ARRAY of at least 8 samplers;
- Sampling gate array with samples less than 30 ps apart;
- RTD/sampler array that triggers with sample spacing less than 50 ps.

RTD-Sampler

The 2 ps sampling optical oscilloscope is an excellent demonstration vehicle for the first building block integrated circuit to be developed for this program: the RTD/ sampler IC. This integrated circuit uses an RTD to generate a 1.2 V positive step waveform that is increased to 2.1 volts by the planar transformer. The transformer then provides balanced positive and negative >1V pulses to the 4-diode sampling bridge. The bridge consists of GaAs Schottky diodes that require 1.6V per pair to provide an on-resistance less than 25 Ω. The remaining >.4V is dropped across the 200 Ω output impedance of the transformer for a diode strobe current of >2 mA.

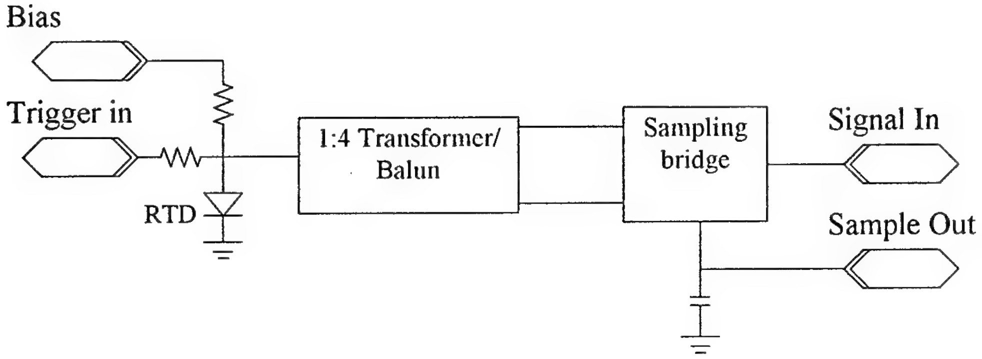


Figure 4: First building block: RTD-sampler IC

With a rise time of 3ps and 2.1 volt swing, the RTD is not preferred over the NLTL for driving the sampling gate in the single-channel sampling oscilloscope. The sampling scope system provides an excellent test set-up for the RTD-sampler. Once the performance of the various RTD-sampler designs is evaluated using the system of Figure 2 and a good design is established, then the multi-channel optical oscilloscope can be developed. The RTD sampler is an enabling technology for this multi-channel ps sampler.

6 ps sampling oscilloscope ARRAY

While the single channel optical oscilloscope described in the previous section achieves a 5x improvement in oscilloscope bandwidth over existing products at low cost, the expense would scale with the number of channels becoming extremely expensive and cumbersome at 32 channels if not for the RTD-sampler ARRAY. This integrated circuit leverages the small size <100 μm^2 and bistability of the RTD to make a multichannel sampler at roughly the same level of complexity and cost as a single-channel sampler based on the NLTL technology. A block diagram of this system is shown in Figure 5. Each sampler can be strobed on successive clock cycles or the samplers can be strobed at a shorter fixed time delay relative to the preceding one. The latter mode of operation sets the stage for a transient capture system where each sampler represents one time bin.

The input to the sampler array as in the single channel version can be optical or electrical or both with a feedthrough design. The optical approach can provide a major advantage for parallel optical link test because GaAs detectors are sensitive to the wavelengths used in the emerging technology and can be monolithically integrated with the samplers. Monolithic integration at the proper spacing makes possible direct ribbon-fiber compatibility. By allowing the user to plug the ribbon fiber directly into the array, reproducibility is greatly enhanced since it is not necessary to separately detect, amplify, and route each channel to a separate sampler.

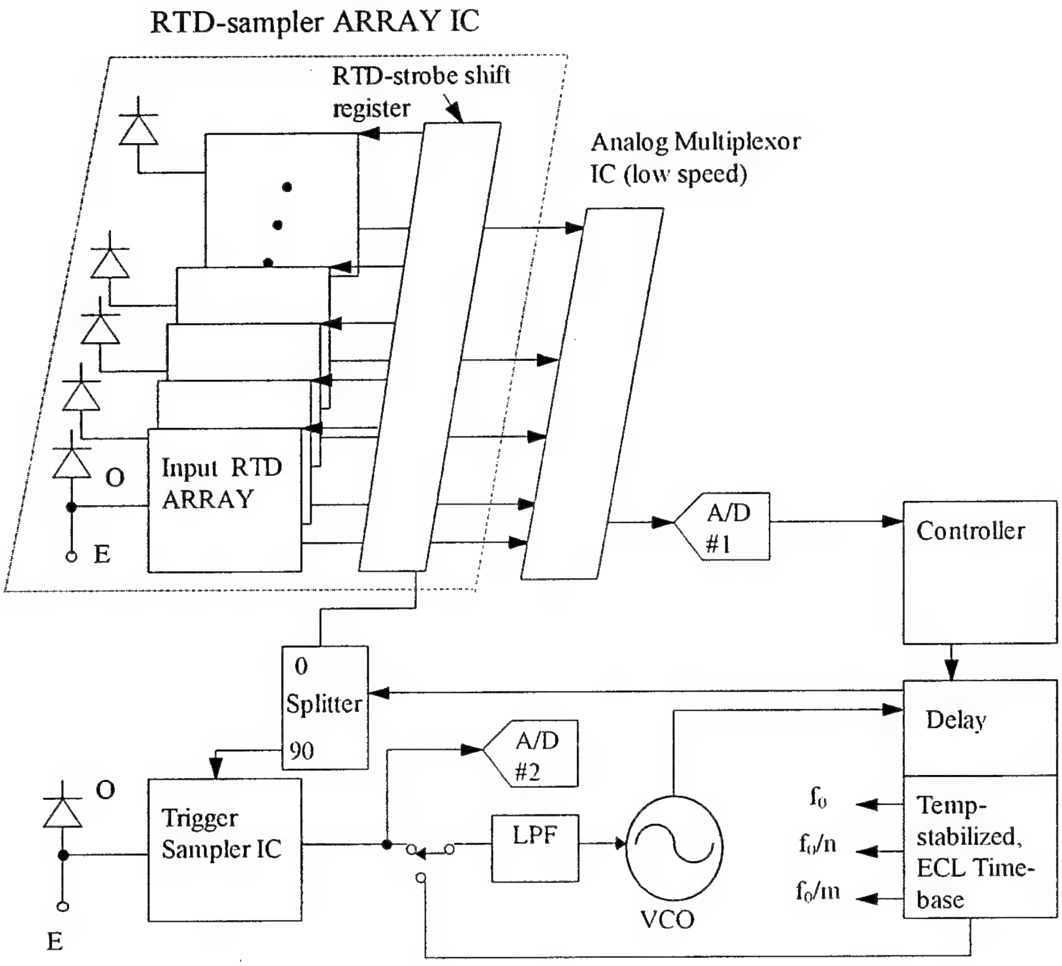


Figure 5: Block diagram of approach for achieving multi-channel sampling oscilloscope for parallel fiber link testing.

30 Gigasample/s transient recorder

Sampling oscilloscopes, boxcar integrators, and the "optical sampling oscilloscope" discussed above, require repetitive input signals. There are a variety of fast phenomena in which the signals of interest are single-shot events of a few picoseconds to tens of picoseconds duration. Laser fusion experiments are a prime example. In other cases, many involving Q-switched lasers, the events are repetitive but occur too infrequently to acquire by repetitive sampling. For signals with bandwidths above ~2-3 GHz, streak cameras remain the primary instrument for observation. The community would benefit from much less expensive instruments. Construction of a complete transient recorder is beyond the scope of this work. Our objective here is to develop RTD ICs as a spin-off of this work that would enable the development of such an instrument.

Specification of performance

A transient digitizer of sufficiently high bandwidth would supplant streak cameras in many applications. A transient digitizer is simply an analog-digital converter (ADC) combined with digital storage and with a trigger system which controls data recording.

Feasible ADC bandwidth is a function of resolution, and resolution requires definition. Frequency response characteristics of an ADC system are not part of the resolution definition (otherwise a 1 GHz, 8 bit converter would require 1GHz•2^8=256 GHz bandwidth). Nonlinearities and distortion from the sample-hold circuit and converter *will* generate distortion components. In applications when small spurious spectral lines are important (ADCs for radar, radio, and some imaging applications), the ADC performance metric is intermodulation distortion, and the effective number of bits of resolution is roughly 1 bit per 6 dB of signal/distortion ratio at a full-scale input. For other applications more characteristic of transient waveform capture, it is the finite ADC quantization which is of concern, and the full-scale signal/quantization noise ratio is the relevant parameter.

We are proposing to develop 30 Gigasample/second interleaved ADCs. Interleaved ADCs employ parallel banks of converters for increased conversion rates; in these ADCs, small interchannel sensitivity mismatches between channels results in a spurious signal. Viewing the sensitivities of the N channels of the interleaved ADC as a series of N numbers at a time separation corresponding to the sample rate, the ADC output is the product of Vin(t) with this series. The ADC output spectrum therefore contains a spurious term whose spectrum is the convolution of the spectrum of Vin(t) with the Fourier transform of the series of sensitivity coefficient mismatches. This "distortion" is a form of modulation noise, the spurious responses are a fixed percentage of the input signal, and there is no noise floor giving rise to a minimum detectable signal. As with the effects of oscillator phase noise in radar, the primary impact of ADC interchannel mismatch is masking of a low-level spectral line of interest by the spurious modulation products of a second, stronger input signal. Effects of channel imbalance can be reduced by calibrating the sensitivity of each ADC channel.

Defining goals for the transient digitizer intended as a streak camera replacement, we seek a 15 GHz analog bandwidth (30 Gigasamples/second), a record length of 500 points, and 10-bits resolution defined in terms of signal to quantization noise. An uncorrected 1%

channel-channel mismatch would result in a 40 dB (6.5 bits) signal/(modulation noise) ratio. With such an instrument we would be able to measure the waveforms of optical transients over a ~60 dB linear input dynamic range. A subsequent Fourier transform of the recorded transient data would contain spurious spectral lines whose amplitude is at least 40 dB below the signal's strongest spectral line.

Primary Specification: 30 GS/s, 500 points, 10 bits (signal/quantization noise), 6.5 bits (signal/channel mismatch) and (signal/distortion)

Table 4: Classification of ADC impairments

Impairment	Magnitude	Impact
signal to quantization noise	$\sim(V_{in})^0$	minimum observable signal
signal to modulation noise (channel-mismatch)	$\sim(V_{in})^1$	masking of weak spectral inputs by strong
signal to distortion ratio	$\sim(V_{in})^2$ and higher	masking of weak spectral inputs by strong

Implementation

A possible implementation of the digitizer block diagram is shown in Figure 6 below. With a required resolution of 10 bits, direct implementation of a 30 GS/s ADC is not feasible. A parallel array of low-speed, high resolution ADCs are instead employed. An input signal is progressively divided into 2^n parallel signal paths. Each signal path is terminated in a track-hold (or sample-hold) IC, whose output drives a slow ADC. As shown, timing for the sample-hold array is derived from a shift register (the array of D flip-flops) clocked at the sampling frequency, with the shift register input driven by a trigger signal.

Transient Digitizer Block Diagram

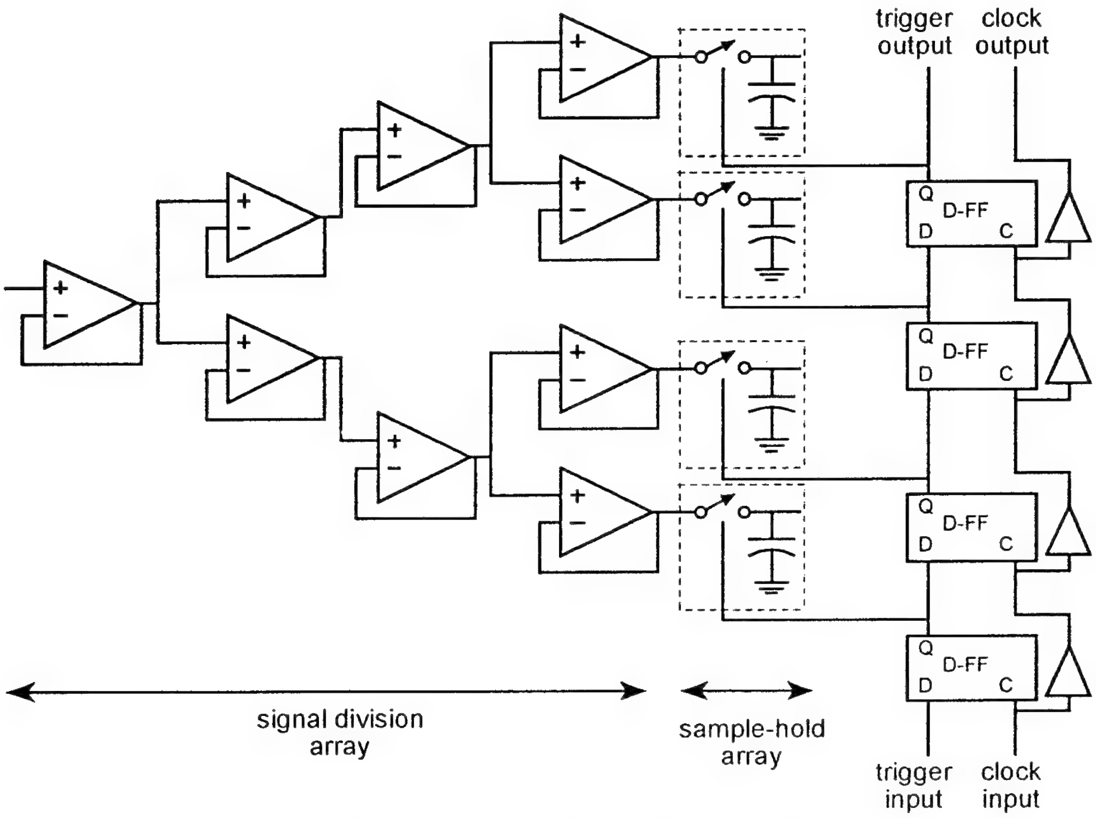


Figure 6: Possible approach for digitizer implementation

The basic building blocks of this system include the RTD/ Sampling gate as previously discussed except now in array form. Additionally, a shift register is required to provide strobe timing and buffer amplifiers to accurately split the signal. The shift register can be implemented in RTD logic and integrated with the sampler array to minimize off-chip connections. However, the amplifier can not be fabricated with the RTD in the processes to be developed here and so must be on a separate IC. Where less accuracy and less sensitivity are required, the inputs of an array can be bussed together, reducing the number of interconnections and required amplifiers by the number of elements in the array.

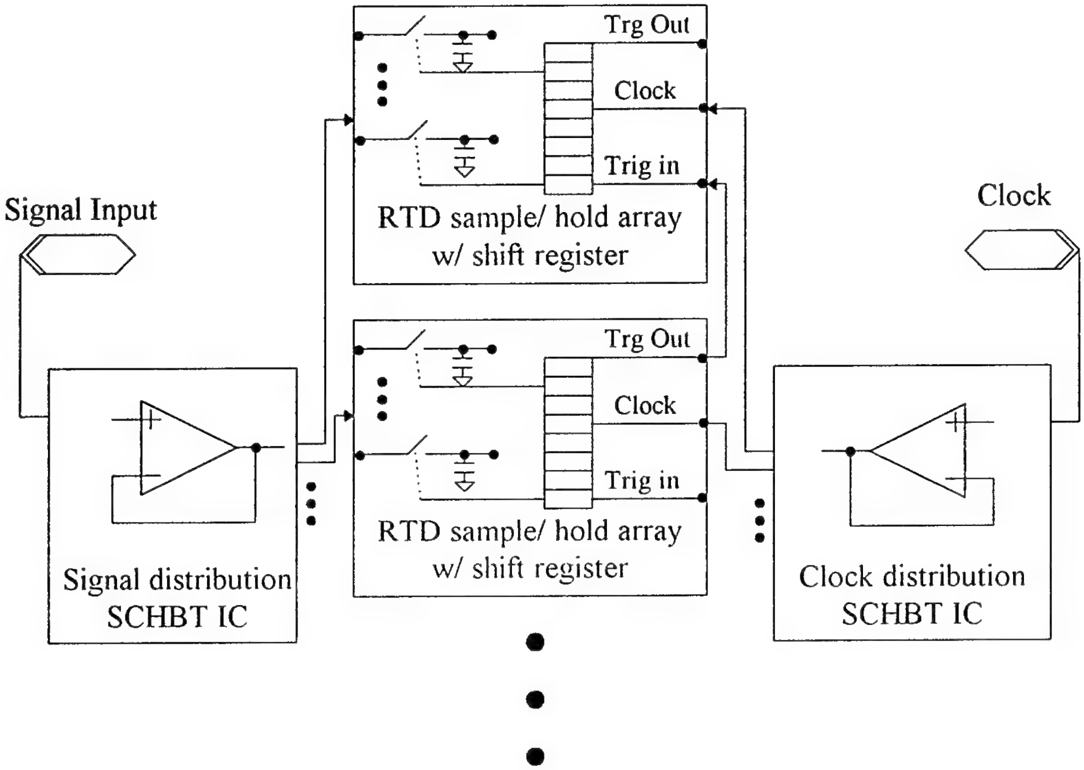


Figure 7: IC hierarchy for implementation of large ADC arrays. The various boxes show the functionality that is expected to be monolithically integrated.

Digitizer Operation:

For applications where 10 bits (signal/quantization noise) resolution is required, the available ADCs have at most 50 Megasample-second conversion rates. A 512 channel array (Figure 7) at 30 GS/s produces 58 MS/s output rates at each sample-hold output. Such an ADC system could be clocked continuously, allowing very long sample records, but today the cost of 512 units of a 10-bit, 50 MS/s ADC would be prohibitive.

Instead, for transient capture, the ADC system does not run continuously. Upon a trigger event, the shift register sequentially loads the 512 sample-hold gates. The sample-hold array is not reset for several tens of microseconds. The 10-bit ADCs need have sample rates of ~100 kHz, and are inexpensive. The maximum record length is equal to the number of channels. The machine cannot run continuously, and must be reset between record acquisitions.

The IC architecture of Figure 7 is general-purpose by design, and a number of different digitizers can be implemented. Given an ~ 8-bit specification, commercial 1 GS/s converters are available, and a 32-channel implementation in the form of Figure 7 would result in a 32-channel, 6-8 bit continuously-running converter having record lengths constrained only by memory depth and memory bandwidth. A highly parallel multichannel boxcar integrator can be similarly realized.

Results and Discussion

RTD Development

This section addresses the results obtained on the following subjects:

- Growth Considerations
- Fabrication Process
- Quantum Well Thickness
- Thickness of barriers
- Doping of Emitter Layer
- Reproducibility
- Pulsed Doping Structures
- RTD Fabrication Process on SI Substrate

1) Growth Considerations

One of the important parameters in the morphology of heterostucture interfaces is the substrate orientation [9,10]. In order to study the effect of substrate misorientation on smoothness of different interfaces between the InGaAs quantum well and AlGaAs barriers, several structures were grown on 0°, 0.5°, and 2° miscut substrates. The growth was stopped at the different layers of double-barrier-quantum-well structure and AFM images were obtained from each of these surfaces. The AFM pictures showed that the surface roughness of the top barrier drastically increased with the substrate miscut. The average surface roughness of the top barrier increased from 3 Å for a singular (0° miscut) substrate to 8 Å for 0.5° miscut substrates. These results are fully discussed in ref. [10]. Also a 15-second interruption time after the growth of quantum well which allows the InGaAs layer to anneal at substrate temperature, can decrease the interface roughness too. The following TEM images, Figure 8 a and b, clearly show that the structures grown with a 15 seconds pause have a more abrupt interface between the quantum well and the top barrier.

[9] M. Suhara et al. J. Crystal Growth, Vol. 179, pp. 18-25,1997.

[10] J. Li et al. The 9th Int. Con. on MOVCD, June 1998.

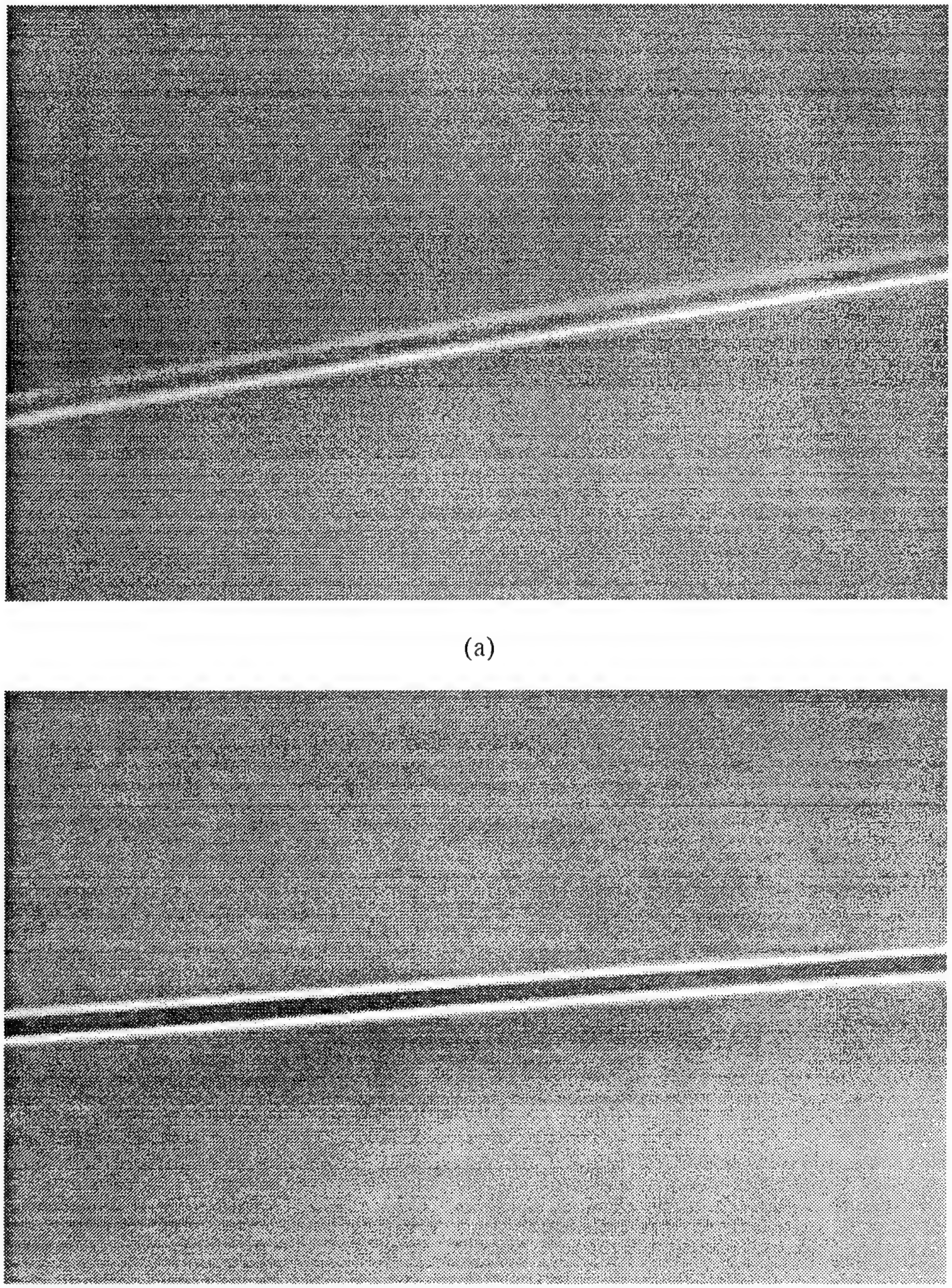


Figure 8: TEM images of AlGaAs/InGaAs/AlGaAs: ~20/50/20 structures, a) with no growth interruption, b) with a 15-second interruption.

2) Fabrication Process

In order to provide a larger top ohmic contact for the device and thus reducing the power dissipation per unit area of the contact, the fabrication process has to be changed to a self-aligned one (i.e. no alignment between the top contact and mesa). In this process the top ohmic contact is used as a mask to etch the mesa. First the top metal contact is deposited by e-beam metal evaporation and patterned by liftoff. Then the bottom metal contact is deposited. The contacts are annealed at 400 °C for 30 seconds. A solution of $NH_4OH:H_2O_2:H_2O$ (1:1:10) is used to etch the mesa at a rate of ~1800 Å/s at room temperature. Ammonium hydroxide has been chosen as the etchant since it provides a fairly isotropic etch for GaAs and does not attack the contact metal. Then the sample is covered by a 2000-Å-thick silicon nitride layer. A thin photoresist is spun over the sample; the thickness of the photoresist is around 0.3 μm and it does not cover the top of the mesas completely. Using Reactive Ion Etching technique, a combination of CF4 and O2 (45 and 10 sccm) is used to etch the silicon nitride from the top of the mesas. Then devices are dipped in OXY35 solution for 10 seconds to make sure that the nitride is completely removed from the top contacts. After removing the rest of photoresist, an overlay metal (Ti/Au 500/5000Å) is deposited and patterned to ~1×1 mm pads. Figure 9 shows a cross section of the fabricated devices.

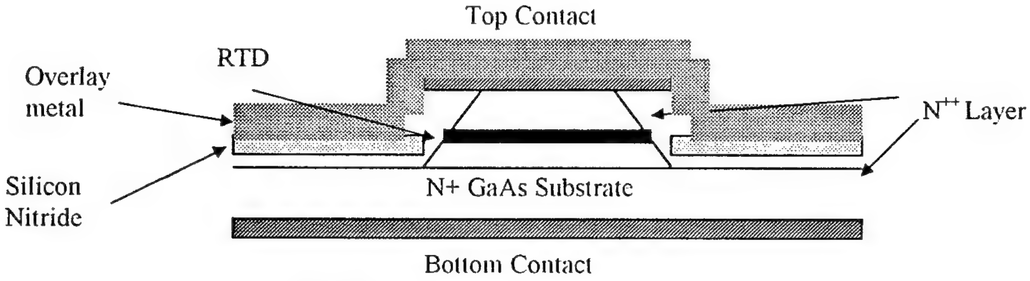


Figure 9: A cross-sectional view of the fabricated devices.

3) Quantum Well Thickness

Using the simulation program, the effects of quantum well thickness on the performance of RTDs has been explained in reference 3. In order to find the optimized well thickness, the following three structures were grown and processed:

Structure A: 30/55/30/200 Å

Structure B: 30/58/30/200 Å

Structure C: 30/61/30/200 Å

All the structures were grown on singular substrates and a 15-second pause was inserted between the growth of $In_{0.3}Ga_{0.7}As$ and the upper $Al_{0.8}Ga_{0.2}As$ layer. The doping of the emitter and collector sides layers was adjusted to 2×10^{18} /cm^3. Figure 11 shows the evolution of RTDs' I-V curve with the quantum well thickness for nominally 15×15 μm^2 devices.

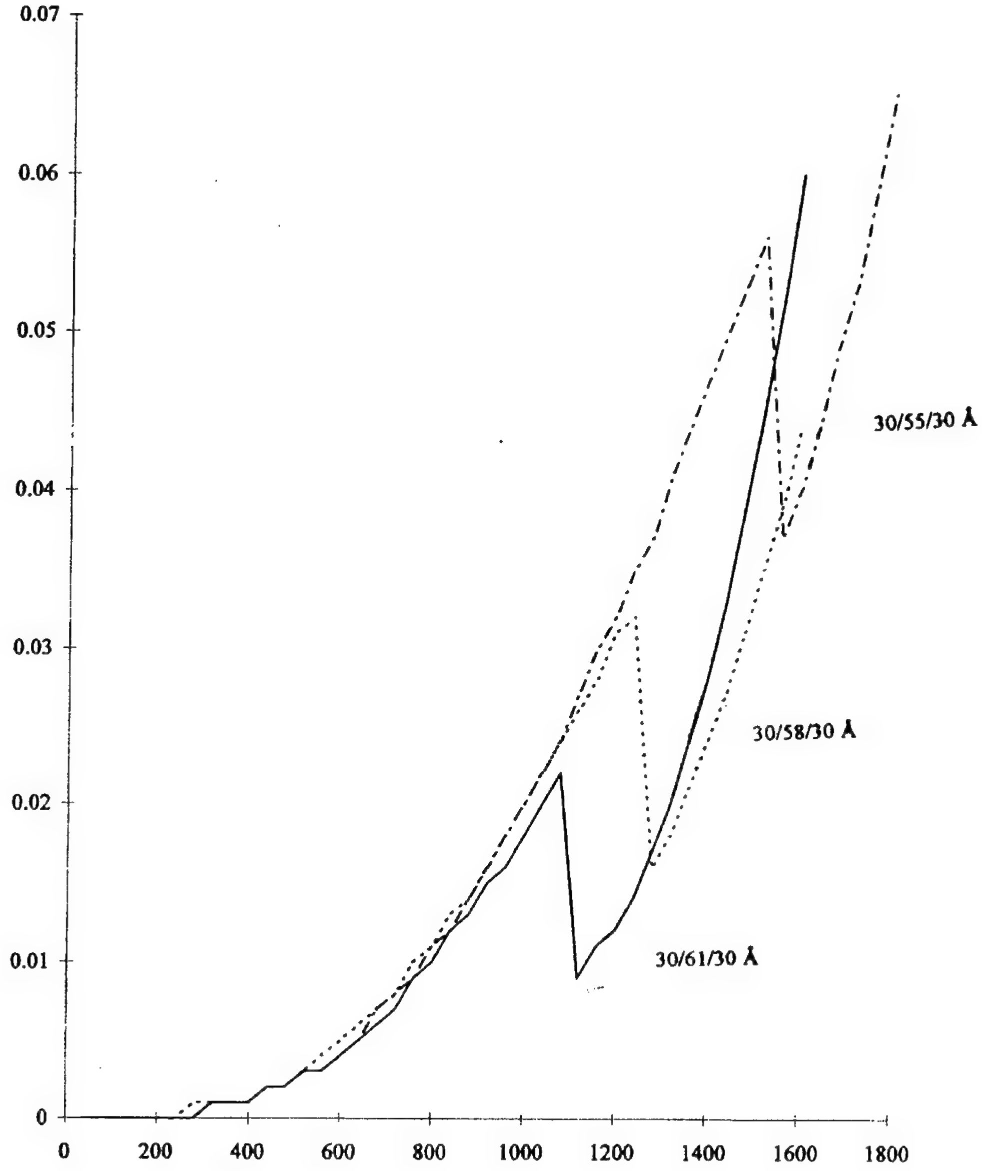


Figure 10: The evolution of I-V curves with the quantum well thickness

Figure 10 shows that by increasing the well thickness and lowering the second energy level, peak voltage and PCD will decrease. Figure 11 shows the exponential-like dependence of PCD with quantum well thickness in agreement with other published results [11].

[11] A. R. Mirabedini et al. Appl. Phys. Lett. Vol. 70, No. 21, pp. 2867-2869, 1997.

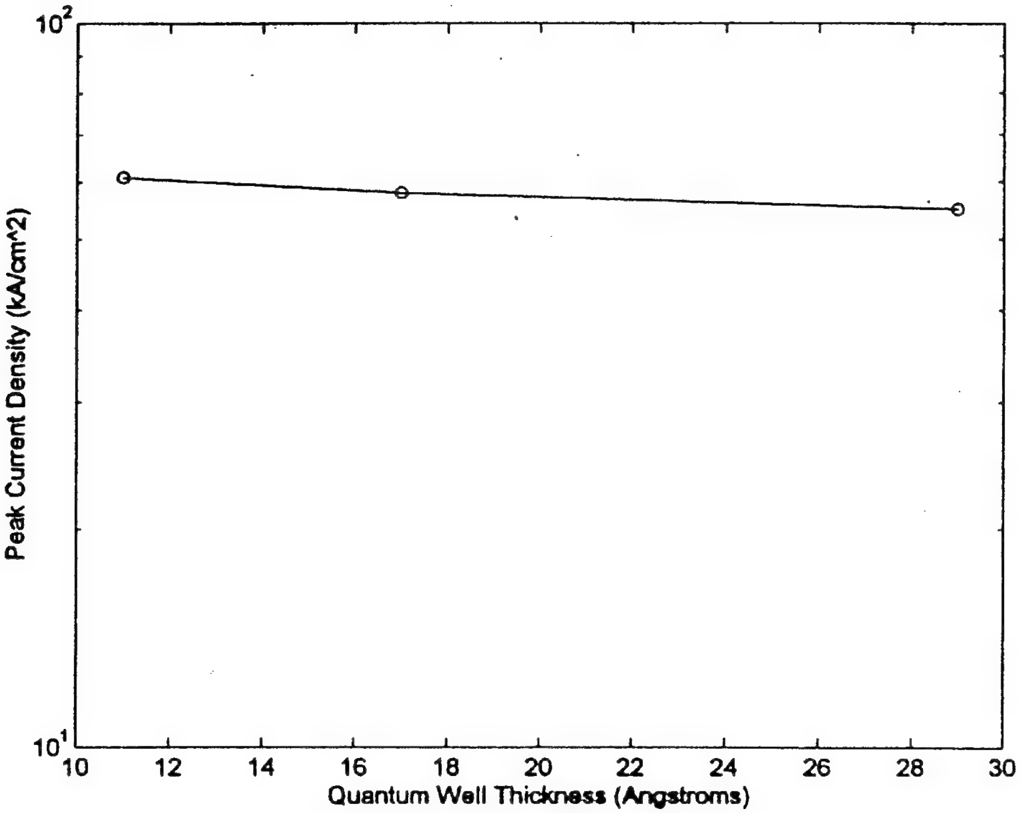


Figure 11: Peak current density as a function of quantum well thickness.

4) Thickness of Barriers

The peak current density of a resonant tunneling diode is an exponential function of barriers' thickness. Refs. [12, 13] report that by adding a monolayer (~ 3 Å) to the emitter side barrier the PCD is reduced by a factor of 56% in structures based on tunneling through the first energy level. To investigate the sensitivity of our structure to the barriers' thickness, the following three structures were grown:

Structure A: 30/58/30/200 Å

Structure B: 24/58/24/200 Å

Structure C: 21/61/21/200 Å

[12] L. L. Chang et al. Editors, "Resonant Tunneling in Semiconductors," NATO ASI Series, Series B: Vol. 277, pp. 71-83, 1990.

[13] T.P.E. Broekaert et al. J. Appl. Phys., 68 (8), pp. 4310-4312, Oct. 15, 1990.

Figure 12 shows the exponential dependence of PCD values to the barriers' thickness. the slope shows that for every monolayer increase (~3 Å) in one of barriers thickness, the peak current density is reduced by 35%. This result shows that the sensitivity of our structure (based on tunneling through the second level) is 60% less than the one for conventional structures.

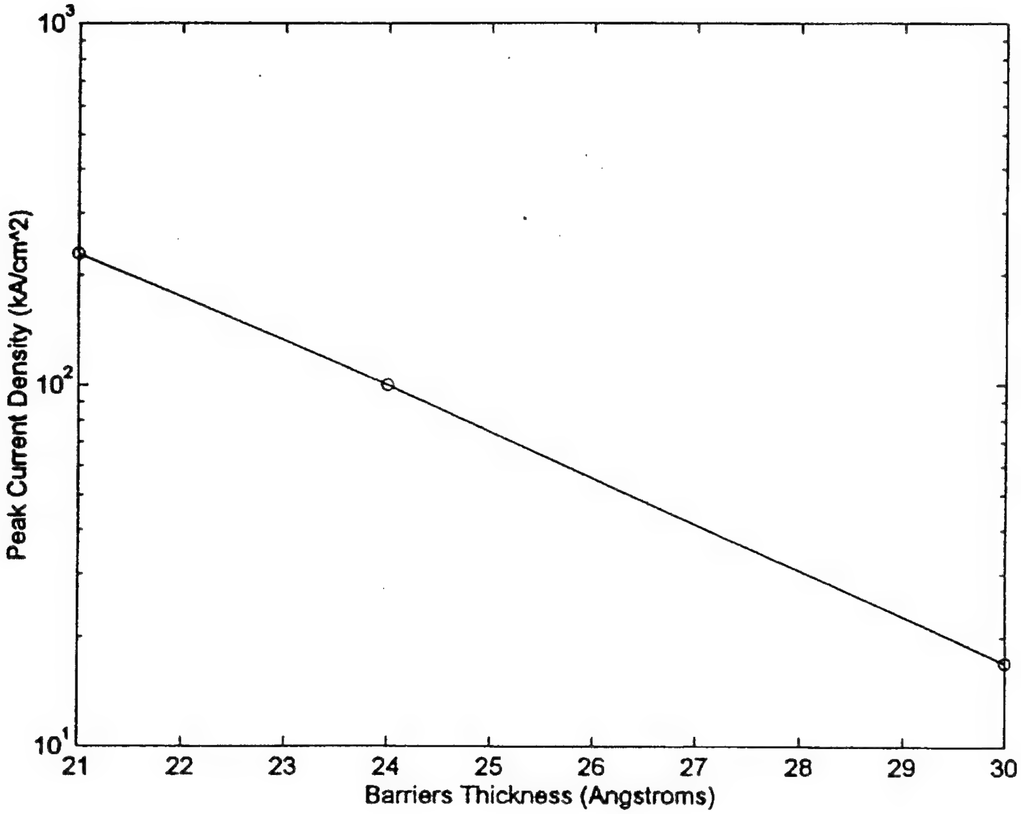


Figure 12: Peak current density as a function of barriers thickness.

5) Doping of Emitter Layer

Based on the above data, the structure $Al_{0.8}Ga_{0.2}As/In_{0.3}Ga_{0.7}As/\ Al_{0.8}Ga_{0.2}As$ /spacer layer: 24/58/24/200 Å was chosen as our optimized structure. The peak current density of this structure is around 100 kA/cm^2 at a peak voltage of 1.4 volts. Figure 13 shows the I-V curve of such structure for a nominally 5×5 μm device. In order to increase the PCD value of the device and reduce the peak voltage to meet the requirements stated in research proposal, the doping of the emitter layer has been increased from $N^{+}=2\times10^{18}$ to $N^{++}=4\times10^{18}$. As expected the PCD increased by a factor of 2 and peak voltage reduced to less than 1.2 volts. Figure 14 shows the I-V curve of such a device.

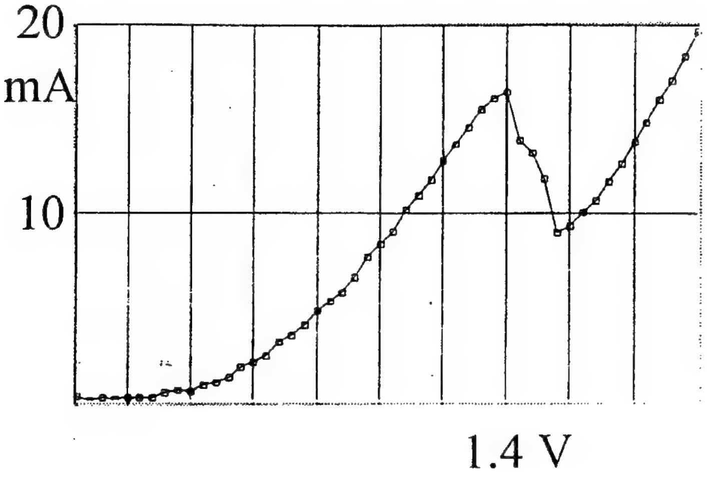


Figure 13: A typical I-V curve for: Al0.8Ga0.2As/In0.3Ga0.7As/ Al0.8Ga0.2As /spacer layer: 24/58/24/200 Å, N+=2x10^{18} ,structure.

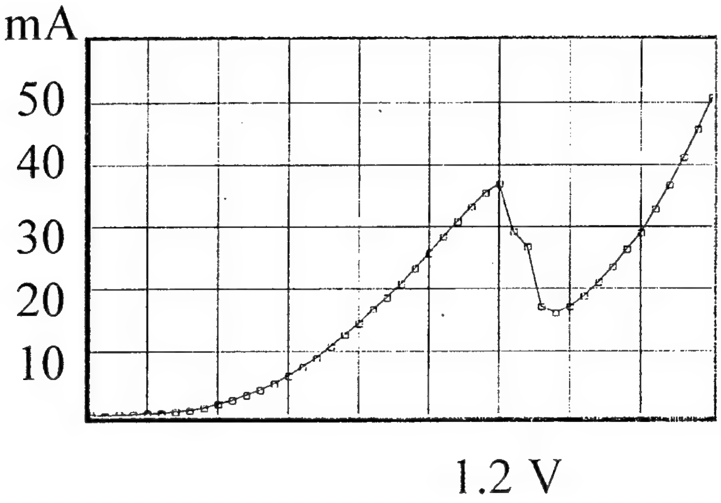


Figure 14: A typical I-V curve for: Al0.8Ga0.2As/In0.3Ga0.7As/ Al0.8Ga0.2As /spacer layer: 24/58/24/200 Å, N++=4x10^{18}, structure.

6) Reproducibility

The reproducibility of high PCD RTDs has been a major concern for the commercial applications of this device. The report by a leading research group in TI indicates that it is extremely difficult to grow RTD structures with reproducible current-voltage characteristics over a longer period of time. Based on this report [14], for two nominally identical RTDs grown 3 months apart, the PCD value changes by a factor of 250%. However, based on this design, two structures grown 2.5 months apart shows only 15% change in PCD value at the same peak voltage. Figure 15 shows the typical I-V curves of two structures grown 2.5 months apart. This superior reproducibility can be attributed to the following factors:

a) As discussed in section 4, a lower sensitivity of PCD to the barriers' thickness, makes our design more tolerant to the fluctuations of barriers' thickness.
b) It has been shown [9] that for quantum wells below 50 Å, the fluctuations in the well thickness is the major cause of band broadening in RTDs, which in turn results in changes in PCD and PVR of the device. Our structure by taking advantage of a wide quantum well is more immune to this fluctuations.

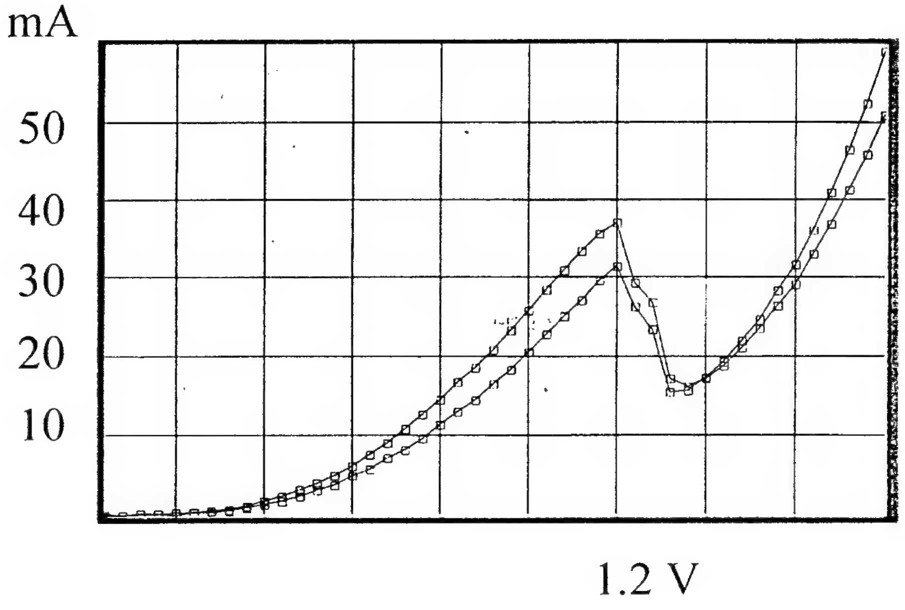


Figure 15: Typical I-V curves of two devices grown 2.5 month apart.

[14] T. S. Moise et al. J. Appl. Phys. Vol. 78, No. 10, pp. 6305-6317, 1995.

7) Pulsed Doping Structures

In order to increase the voltage swing of the device in a pulse-forming application, one of the following methods can be used:

a) By using a 100- Å thick pulsed doping layer (N ~ 2×10^{18}) sandwiched between two undoped regions (200 and 400-Å thick), it is possible to increase the voltage swing and reduce the device capacitance at the same time [15].

b) It has been shown that by using an InGaAs prewell (~ 45 Å) immediately before the first barrier, it is possible to improve the device PVR, which in turn increases the voltage swing [16].

c) The typical PVR for GaAs/AlAs structures is >3. However, the typical PVR value for GaAs/AlGaAs devices is around 2. By increasing the barriers height and reducing the thermionic emission, a better PVR will be attainable.

The following delta doping structure was grown:
$Al_{0.8}Ga_{0.2}As/In_{0.3}Ga_{0.7}As/$ $Al_{0.8}Ga_{0.2}As$ /spacer layer 1/ delta-doping layer/ spacer layer 2: 24/58/24/200/100/400 Å.

Based on simulation results the doping of pulsed doping layer was chosen as: $N^{++}=2.6\times10^{18}$ cm^{-3}. Figure 16 shows the typical I-V curves for similar structures a) without and b) with pulsed doping layer. The pulsed doping structure shows lower sensitivity to thermionic emission for voltages greater than the valley voltage, and thus improves the voltage swing of the device by a factor of 1.3.

[15] L. Yang et al. IEEE J. Solid-State Circuits, Vol. 29, No. 5, pp. 585-595, 1994.

[16] S. Lee et al. IEEE Trans. Electron Devices. Vol. 36. No. 11. pp. 2619, Nov. 1989.

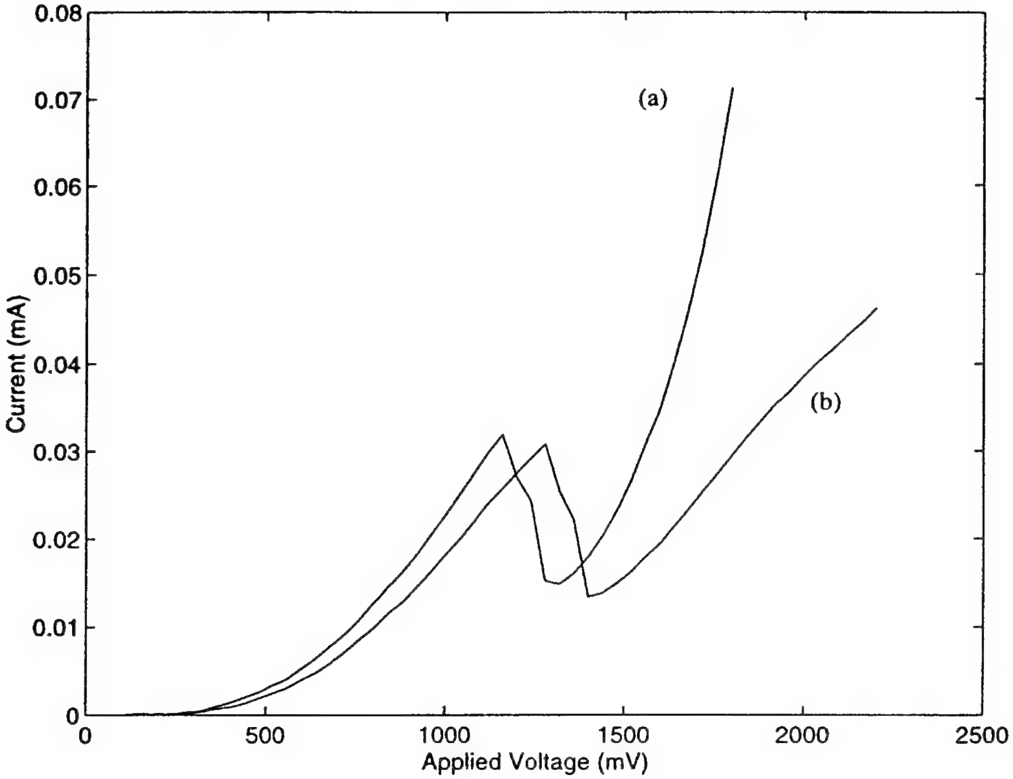


Figure 16: Typical I-V curves for structures a) without and b) with pulsed doping layer.

8) RTD Fabrication Process on SI Substrates

Two major techniques for device isolation in III-V compound substrates are etch and proton isolation. The etch technique results in a nonplanar structure, which makes further device interconnection a difficult task to do. Also this technique imposes a high minimum feature for circuit integration. On the other hand, proton isolation leaves us with a planar structure, which lends itself to further integration. Literature reports [16, 17] show that by adjusting the dose and energy of the incident ions, it is possible to achieve good isolation n GaAs substrates ranging from 2 MΩ/□ to 10 MΩ/□. On the other hand, InP substrates do not respond very well to proton isolation.

Fabrication process consists of four lithography steps (4 masks). First step is to deposit and pattern the top ohmic contact (mask 1). Top ohmic contacts are 3-7 µm wide (limited by the etch process) and 5 µm long (limited by the proton isolation process). Second step is to define the active device area longitudinally by proton isolation. In proton isolation process, in order to save the active areas from proton damage, a thick layer of gold (1.6 µm) on top of those areas is necessary (mask 2). After implantation process, Au mask has to be removed, so a sacrificial layer of polyimide needed to be placed underneath it. Polyimide baked at 240 °C is hard enough to provide a base for subsequent lithography

[17] D. C. D'Avanzo, IEEE Trans. Electron Devices, Vol. 29, pp. 1051-1059, July 1982.

and Au deposition, yet this temperature is not high enough to crosslink the polymers completely and so the polyimide is still can be removed. Third step is to etch the structure deep to the bottom N^{++} layer (mask 3). This etch also defines the width of device active area. Without removing the patterned photoresist, a second ohmic metal is deposited and patterned by liftoff to form the self-aligned bottom ohmic contacts. In order to prevent germanium from diffusing on the lateral sides of the structure (the etched sides) during annealing process, the structure is covered with a 2000-Å-thick Si_3N_4. The fifth step is to anneal the contacts. Then it is time to deposit and pattern the interconnect metal (Ti/Au 100/10000 Å) which provides us with the necessary microwave pads to test the device (mask 4). Figure 17 shows a cross-sectional view of the device.

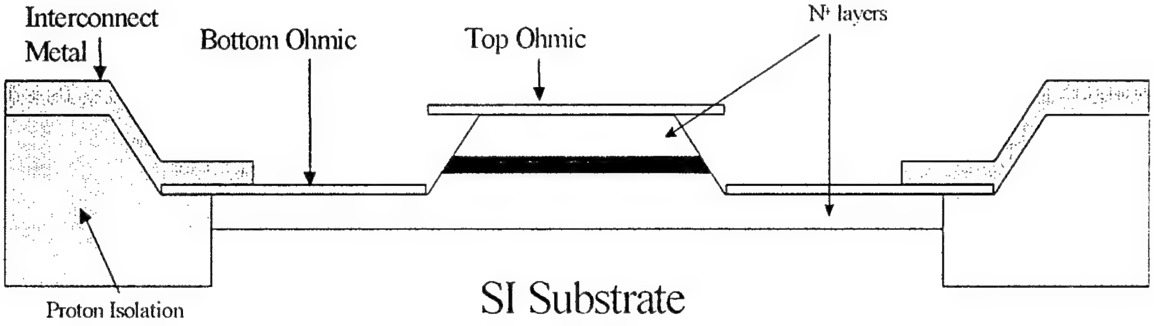


Figure 17: Cross-sectional view of a third generation RTD.

NLTL-based Test-System Development

A sampling oscilloscope with ~2ps risetime is required for testing all of the RTD circuits that are/were developed for this program. Sampling oscilloscopes consist of a sampling device with pulse generation, a triggered timebase, sample storage and data display. Sampling oscilloscope bandwidth is determined by the sampling device and by the timebase. Each of these subsystems, the timebase or Delay System, the Sampling System and the Computer System, along with their particular components will be described in the following sections.

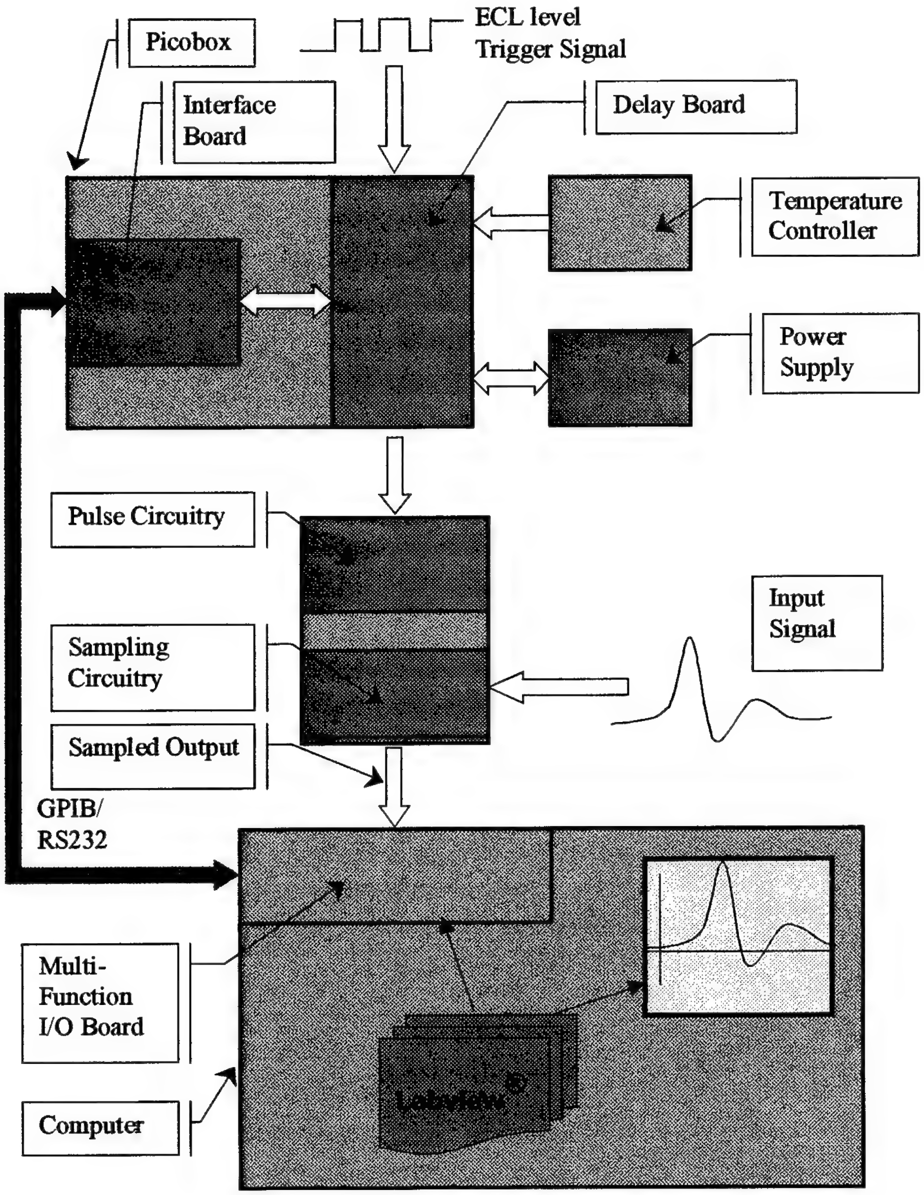
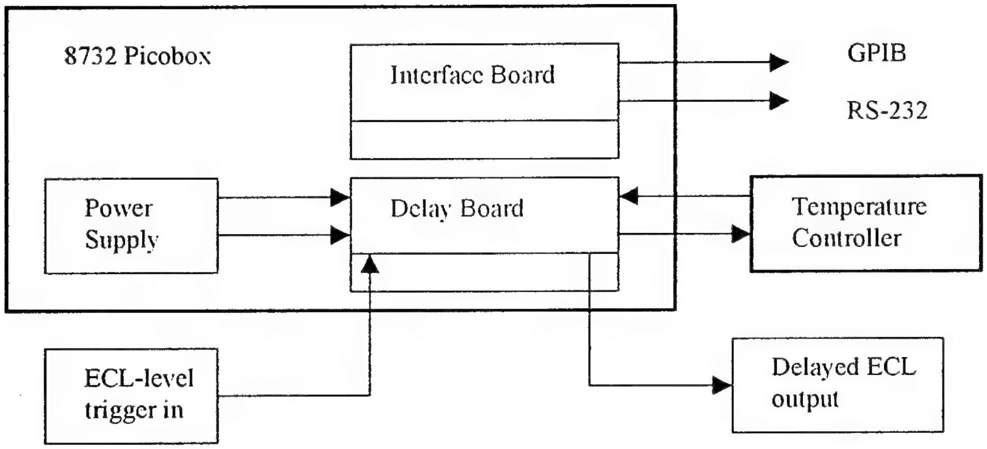


Figure 18: NLTL-based 2-ps sampling oscilloscope test system

Delay System

The Delay system controls the spacing of the samples that are taken of the signal of interest. The output of the delay system is used to drive a pulse generator. The delay system has five main components: Picobox, Interface Board, Delay Board, Temperature Controller and Power Supply. In a typical commercial sampling oscilloscope, the samples are handled independently. A trigger hold-off is used to provide the system processing time before the next trigger is accepted. The maximum trigger rate is less than 1 MHz. In contrast, our present system performs a moving average of the sample data by continuously storing sampled charge and removing it at a slow rate. This type of sampling is sometimes referred to as "slow walkthrough." Triggers are continuously accepted up to a rate of 900 MHz. Millions of samples are accumulated before moving on to the next delay setting. This type of system is actually much easier to implement than the commercial system because no trigger hold-off is required and the built-in averaging reduces the noise requirements for the signal sense circuitry. The disadvantage is that only single-valued (relative to the trigger), deterministic signals can be measured. Pseudo-random data is not readily handled. Our plan is to implement independent sample processing once the more simple system is proven.



Picobox

The Picobox is part of the New Focus 8732 Picomotor driver. It is used as the main development platform for the Delay System. Because of its use as an existing New Focus product, it offers an amount of already completed hardware that would not have to be duplicated in this program. The Interface Board and Delay Board (to be described next) plug into the Picobox on a common backplane. There is a switching power supply that delivers +/- 15VDC and +5VDC for the plug in boards. The Picobox also has a front panel that is used for alphanumeric display and for control of programmed features.

Interface Board

The Interface is a plug-in card for the 8732 Picomotor driver that contains all of the communications hardware, both internal to the 8732 and to the external world. The Interface card is used as the main 'brains' of the Delay System for communications that occur between the computer that is displaying the data and the delay board that is used to sample the signal of interest. The interface board hardware includes a 80C51 type microcontroller, flash memory for program storage, a GPIB interface chip, a RS-232 chip and some miscellaneous glue logic. The existing 8732 firmware was used as a starting point for development of the Delay Box firmware.

SOFTWARE COMMANDS

The existing Interface Board firmware had the ability to address the Delay board but it soon became evident that a more 'scope-like' performance from the Delay system would be needed. Since the development of a totally integrated sampling system was not within the scope of the program, a computer interface and command language were devised. Borrowing from existing industry standards, a SCPI command set that parallels other manufacturers was defined and then implemented in the Interface Board firmware. This allows for the use of the Delay system without an intimate knowledge of it's internal workings; it also increases the overall system performance by reducing the amount of communications that must occur during data acquisition and thus speeding up the measurement. For example, to command a sweep of delay values (which corresponds to a oscilloscope sweep) without the command set, it takes 5*recordlength commands (two for coarse and three for fine delay.) Using the command set it can be done with a single command. The command set is fully described in

Appendix 1 – Command Set.

FIRMWARE

There are three code blocks that were modified or created for this program. The first block (REMOTE.C, REMCMDS.H) controls the user interface to the Delay System. As each user command is parsed, an appropriate routine is called to execute the command. The second block (REMFUNCS.C, REMFUNCS.H) is where each of the user command routines is defined. This methodology followed the already existing code and allows for the quick addition of new commands. The third block (DELAYS.C, CONSTDEF.H) has the routines that are specific to the Delay board and Picobox hardware. Global system definitions are in the file GLOBDEF.H. Approximately 2500 lines of C code were generated for this program. The existing software development system used by New Focus was used for this development. The code files described above are reproduced completely in Appendix 2 – Code Files.

Delay Board

The Delay Board communicates with the Interface Board and varies the delay time of the trigger signal that is input to the Delay Board. The range is 4.5 ns with a resolution of approximately 0.01 picosecond. The delay times must be stable over temperature and voltage variations. According to Motorola documentation on the sensitivity of the 196/195 to temperature and voltage variations, one can derive an equation.

$[(4.7ps/C)*\delta T] + [(3.5ps/V)*\delta V] = \delta t$

where C is degrees Celsius, V is power supply volts. So, for δt better than 0.1ps, δT must be less than 0.01 C and δV must be less than 3 mV. Presently, the voltage stability is being met, while the temperature is stable to within 0.05 C. We expect to improve to 0.01C when the PCBs are modified to allow the box to be sealed providing a more stable thermal environment. With the present temperature stability, overall calibration drift of 0.25 to 0.5 ps is expected worst case.

The board is implemented using PECL (positive ECL) and HC (high-speed CMOS) logic. The trigger signal is ac-coupled to a differential receiver with high gain (approx. 40). The bias level is derived from the power supply described in the next section. The receiver then drives the delay chips (MC100E195 and MC100E196). The output of the delay chips is fed to a pair of output drivers that are ac-coupled to the output SMA connectors. The fine control is driven by an AD7245 12-bit digital to analog converter that outputs a 0V to 5V signal. The Delay board interfaces to the Interface board through the common backplane (a 96 pin DIN connector) using high-speed CMOS logic gates and ECL-TTL level shifters. The schematic of the Delay board is given in Appendix 3 – Delay Board Schematic. The Delay Board was implemented in a four layer printed circuit board of 0.062" overall thickness with controlled impedance of 50 ohms for the trigger signal path. The printed circuit board layout is given in Appendix 4 – Delay Board PCB layout.

Temperature Control

The temperature controller was not designed for this program but was taken from an existing New Focus product, an external-cavity diode laser. This controller, in a similar application, has been used to deliver better than the required regulation of 10 mK. The temperature-sensing element was a AD590 sensor, again the same device used in other New Focus products. This element was embedded within an aluminum block that was affixed to the delay chips. The temperature controller drives a pair of peltier devices from Melcor (FC0.45.66.05L) that are affixed to the top of the aluminum block and cooled with a common 12V CPU fan.

Power Supply Control

The power supply that is used to drive the Delay board is derived from a standard LM723 voltage regulator. This regulator by itself cannot drive the Delay Boards requirements but with an external pass transistor (NPN 2N3054) with proper heat sink, can easily drive the required power levels. The input voltage is +15V and the output voltage is +5V. Though the input-output differential is 10V and hence causes considerable heat generation, the regulation characteristics of this topology give us an expected load regulation of 2mV. A schematic of the power supply is given in Appendix 3 – Delay Board Schematic. The power supply was implemented using discrete components and prototype board techniques.

Sampling System

This system uses the input trigger signal to sample a signal of interest at a rate determined by the trigger signal. A block diagram is shown in Figure 19. Waveforms at points 1, 2, and 3 are shown in Figure 20.

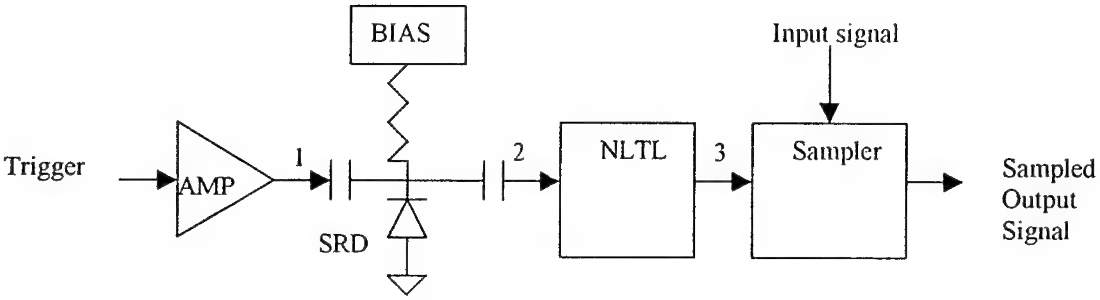


Figure 19: Sampling system block diagrem.

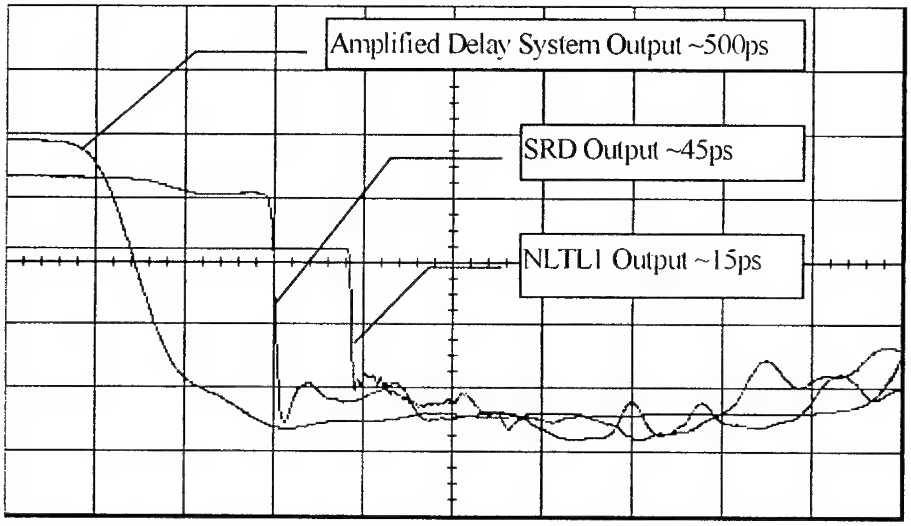


Figure 20: Voltage waveforms at various points in the sampler system. Vertical scale: 2V/ Div, Offset: 0V, Horizontal Scale: 500 ps/ Div. NLTL transition time is limited by the 20 GHz oscilloscope bandwidth.

Amplifier

The amplifier is a Mini-Circuits ZHL-2-12 amplifier with 24 dB min gain. This is used to amplify the ECL level signal from the Delay box (approx. 0.8V pp) to a level that is appropriate for the SRD (approx. 15V). The rise/fall time of the ECL signal is approx. 250ps. This amplifier was chosen for it's bandwidth of 10MHz to 1.2GHz and because of it's high linearity (IP3 of 38dBm).

SRD

The SRD (step recovery diode circuit) is used to compress the edge of the amplified ECL signal down to 100 ps. The diode used is a Metallics MMD-840. The bias level is chosen to allow for complete charging of the SRD at the duty cycle and frequency of the trigger signal.

NLTL

The NLTL (Nonlinear Transmission Lines) pulse generators are in the form of GaAs integrated circuits, containing a microwave transmission line loaded by Schottky diodes. Nonlinear electrical wave propagation on these devices results in the compression of input electrical pulses during propagation to the NLTL output. Signal transitions of several volts amplitude and a few picoseconds duration are readily formed depending on the specific design of the NLTL. These pulses are used to momentarily gate the Schottky-diode sampling bridge. This project used two different monolithic NLTLs, NLTL1 and NLTL2. Their schematic representation is given in Appendix 5 – NLTL Schematics and their physical layout is given in Appendix 6 – NLTL Physical Layout. The NLTL2 is used to give more compression than NLTL1, down to 5ps for NLTL2, 10ps for NLTL1. The NLTL/sampler technology is licensed from Stanford University. The significant distinction is that the devices are fabricated on MOCVD-grown material in anticipation of integration with the MOCVD-grown RTDs. During the first year of this contract devices were fabricated that had sub-par breakdown characteristics. During this second year, additional devices were fabricated with significantly better breakdown voltage as shown in Figure 21. Breakdown voltage is critically important to attaining the full compression of the NLTL.

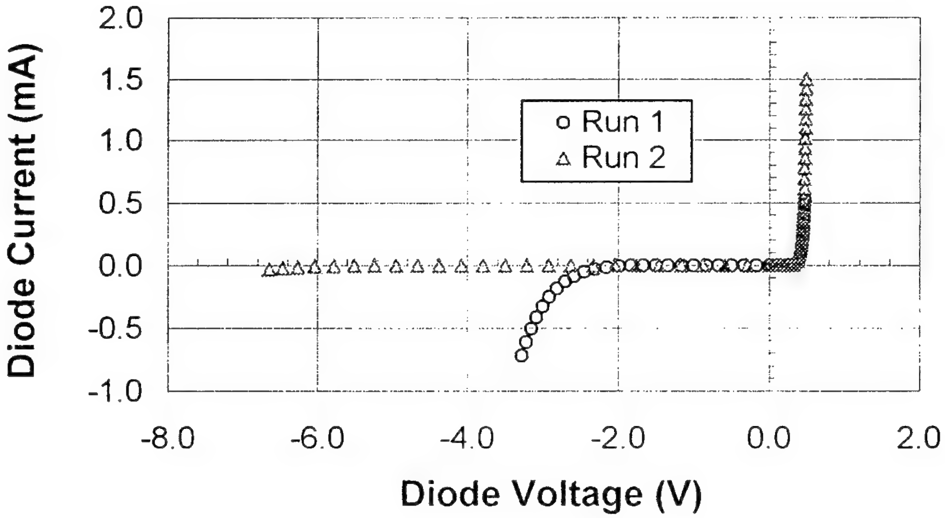


Figure 21: Graph showing the improvement in reverse breakdown achieved with the second fabrication run.

Sampler

The Sampler technology has been well described elsewhere in the literature and for this program the primary goal was for the integration of that technology into a functional test system. A significant design addition was the integration of a GaAs 100 GHz photodiode directly input to the sampler. Their schematic representation is given in Appendix 7 – Sampler Schematic and their physical description is given in Appendix 8 – Sampler Physical Layout.

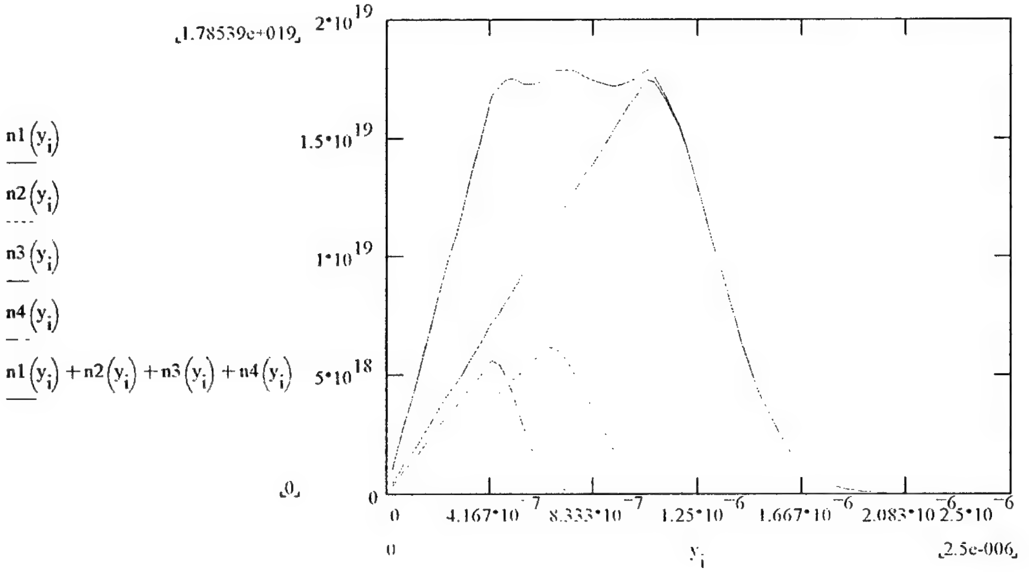
Processing

The process we used for the NLTL and Sampler integrated circuits is a standard GaAs process that has been developed at Focused Research for fabrication of GaAs Schottky photodiodes. A process flow diagram is given in Appendix 9 - Process. In order to accommodate the specific device structure for this program, both MathCad and TRIM simulations were performed to ensure that the isolation step was correctly implemented. A MathCad result is shown below for a given implant schedule (of energies and dosages) of the final implanted proton distribution (vertical axis (cm^{-3})) as a function of depth (horizontal axis (m)).

$Q1 = 5.00 * 10^{14}\ \text{cm}^{-2}$ $\quad E1 = 160$ keV

$Q2 = 1.10 * 10^{14}\ \text{cm}^{-2}$ $\quad E2 = 100$ keV

$Q3 = 0.65 * 10^{14}\ \text{cm}^{-2}$ $\quad E3 = 65$ keV



Microwave Housing

A critical part of this program was the ability to transfer with high fidelity the trigger and input signals. To ensure success we have leveraged off of New Focus's experience to modify existing housings to fit the aspect ratio of the integrated circuits from this program. The final drawings as manufactured are shown in Appendix 10 – Microwave housing drawings.

Computer System

The entire test setup is controlled from a Labview program running on a PC. This allows for quick prototyping of command and control functions that are used in the sampling scope as well as prototyping of functions that are embedded within the delay box. The program controls a multi-function I/O board (with on-board A/D, D/A, and digital I/O) and communications through both GPIB and RS-232.

Multi-function I/O board

This board is a National Instruments MIO-16E. The A/D has 12-bit resolution with sampling speed up to 100,000 samples per second. The A/D also has programmable gains to aid in matching the dynamic range of the A/D with the output signals from the sampler. The digital I/O signals of the board are used for direct communication between the Delay Box and the computer, for example to signal the end of a data acquisition.

GPIB/RS-232 communications

The GPIB communications are accomplished with a National Instruments GPIB card and the RS-232 communications are from the built-in COM ports of the PC. GPIB is used primarily for calibration while RS-232 is used during sampling data acquisition.

Labview Program

Calibration

An important operation in the development of the test system involved the delay calibration. At the sub-picosecond level, considerations must be made for all of the components involved: synthesizers, cables, and oscilloscopes. As shown in Figure 22, one of the techniques for eliminating the short-term jitter in the reference signal source was to trigger the HP scope with the same signal as sent to the time delay generator. The Labview front panel and diagram are shown in Appendix 11 – Labview Calibration.

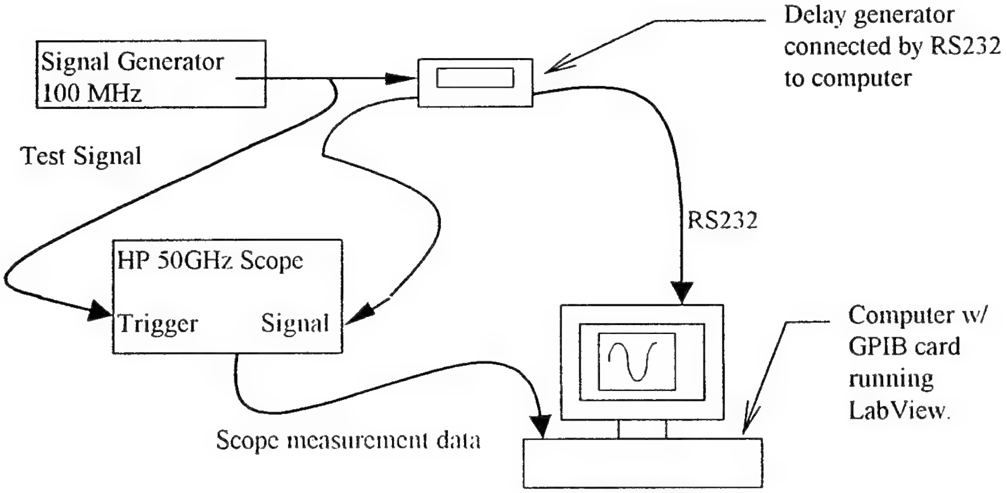


Figure 22: Calibration Setup

Results

After examining the calibration data and recognizing that we were not achieving our goal of sub-picosecond stability, the stability of the signal source, cabling and oscilloscope were called into question. The results of a calibration ***without*** the Delay System are shown in Figure 23. We believe that the long-term rise in drift is from the change in the room ambient temperature. Another test was performed with six-inch cables to better isolate the cause of the shorter-term drift but the results were very similar. These results pointed to the oscilloscope, our time-measurement algorithm, and environmental conditions.

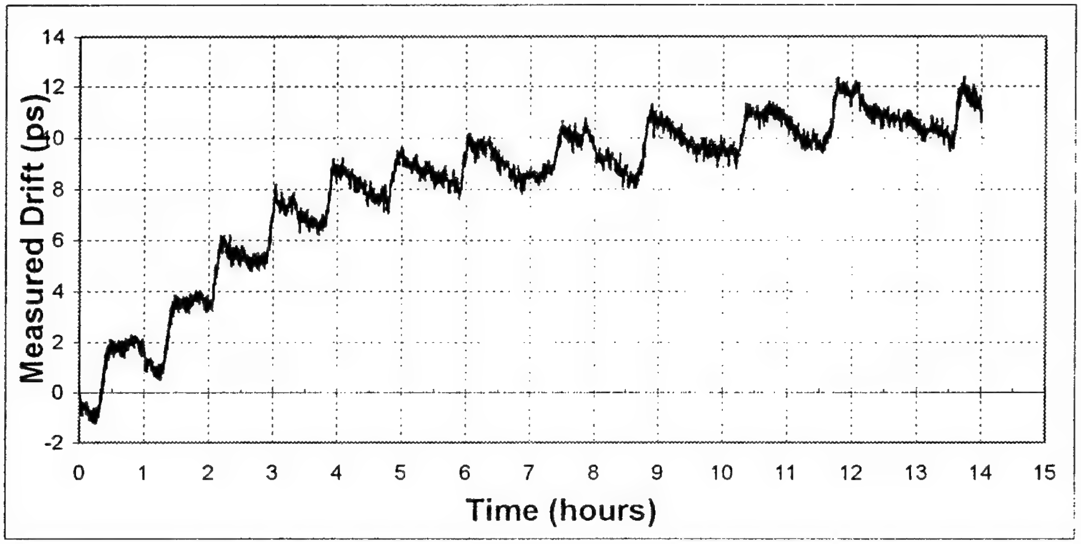


Figure 23: Drift of test equipment used to calibrate delay generator.

While we had made an effort to minimize the calibration time we did not believe that we could reduce it below 8 minutes, the time of the periodic 2-ps jump in measured drift. Because of this, we modified our calibration routines to reestablish the zero reference for every delay measurement. The results are shown in Figure 24 and Figure 25. After further analysis it was found that our zero (or any single point measured repeatedly) had a range of values of 2.5ps with a standard deviation of 0.47ps. To further confirm our results we looked at the correlation between the desired delay and the actual delay measured. The desired delay has both coarse and fine components. As is shown in Figure 26, there seems to be some correlation between increasing delay and the error measured.

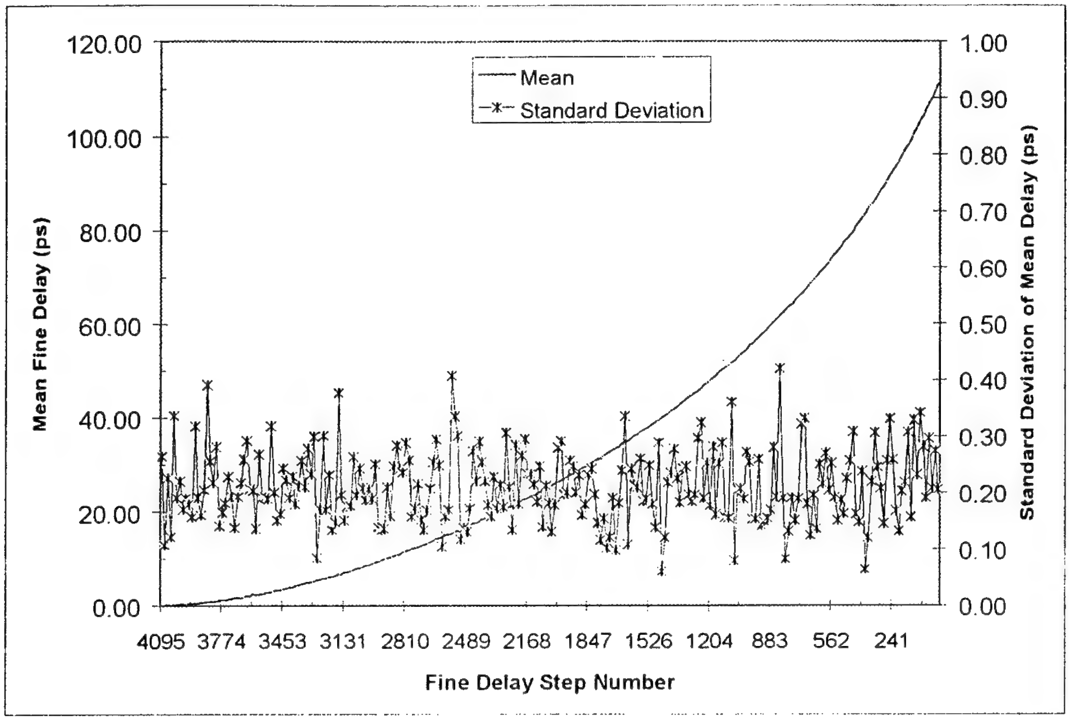


Figure 24: Fine Delay mean delay and standard devication over 8 calibrations

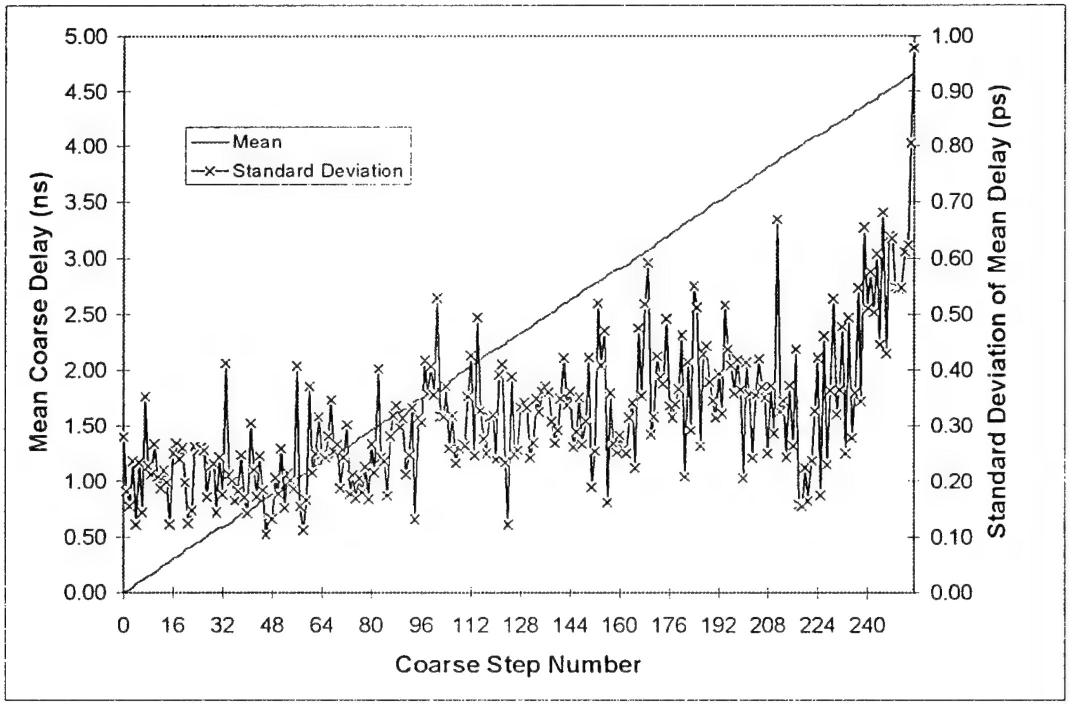


Figure 25: Coarse Delay mean delay and deviation over 8 calibrations

Coefficient Generation

Once calibration data was taken, it was statistically reduced to reduce short-term effects. The entire coarse data set is used and downloaded to the Delay Board. The fine delay is curve fit to a quadratic equation. The coefficients are then downloaded to the Delay Board. The coarse data and fine coefficients are then used to calculate the required delay. The Labview front panel and diagram is given in Appendix 12 – Labview Coefficient Generation.

Data/Coefficient Verification

An important step before use of the calibration data is the verification of the generated calibration data to the expected delay. This is accomplished by exercising the delay across the entire range of values. Standard statistical techniques can then be used to verify the quality of the calibration. The Labview front panel and diagram is given in Appendix 13 – Labview Data Verification.

Download

Because of the development nature of this program it was fortunate that a mechanism for upgrading of the firmware existed in the New Focus 8732 Picomotor. This technique was used and is shown in Appendix 14 – Labview Data Download.

Correlation

Just as it was important to verify the calculated data with the generated data, it is imperative that actual measurements are taken to verify the quality of the delay. A plot of this is shown in Figure 26: Correlation between desired delay and measured delay. The Labview front panel and diagram are shown in Appendix 15 – Labview Correlation.

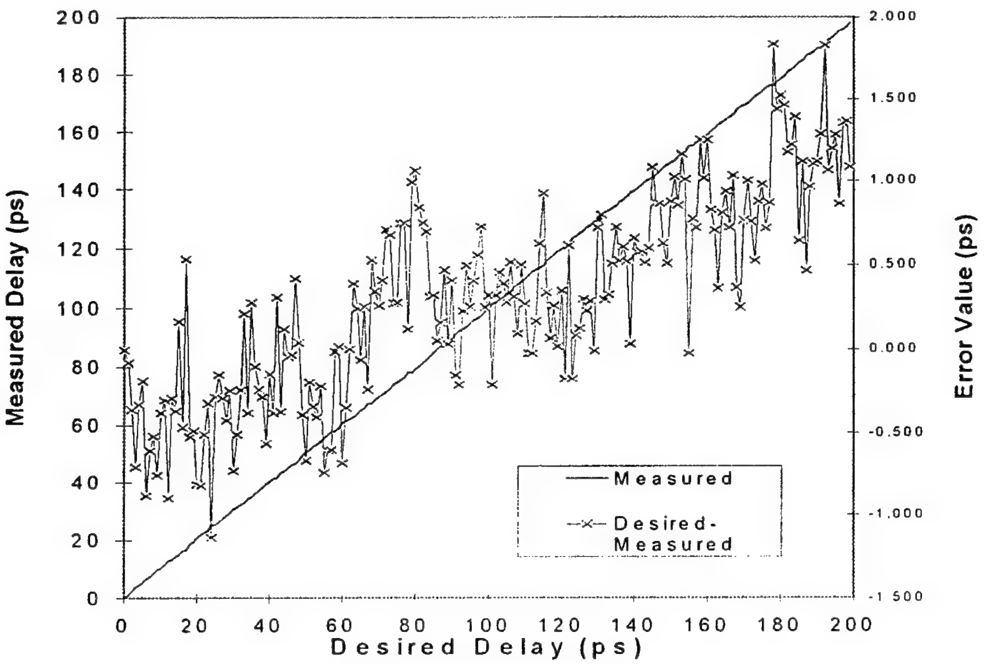


Figure 26: Correlation between desired delay and measured delay. Fluctuations are within the range of experimental error; over all rise in error with increasing delay is probably due to calibration drift from insufficient temperature regulation.

Oscilloscope

The experimental setup for oscilloscope testing is shown in Figure 27. The first attempt to measure a waveform resulted in data with an overall baseline wander and periodic discontinuities. These effects are due to time-base error and trigger-pulse feed-through respectively. Fortunately, the effects are fixed and can be removed manually from the data. We will address the root cause of these problems in the follow-on program. With the fixed structure removed, our sampler can be compared to the HP 54750A scope with 54752A 50 GHz plug-in. This comparison is shown in Figure 28. The record length was 1000 points, with 200 averages per point. While the comparison is not perfect in the amplitude, there is good agreement in the time axis, with the sampling scope following the time response of the HP scope across the whole record length. We believe that the signal distortion is caused by improper sampling bridge operation. The sampler that we had time to package and characterize had one failed hold capacitor. While this alone is not sufficient to cause this type of distortion, it indicates that there is plenty of work to do to fully characterize these circuits. Full characterization is the first task of the follow-on program.

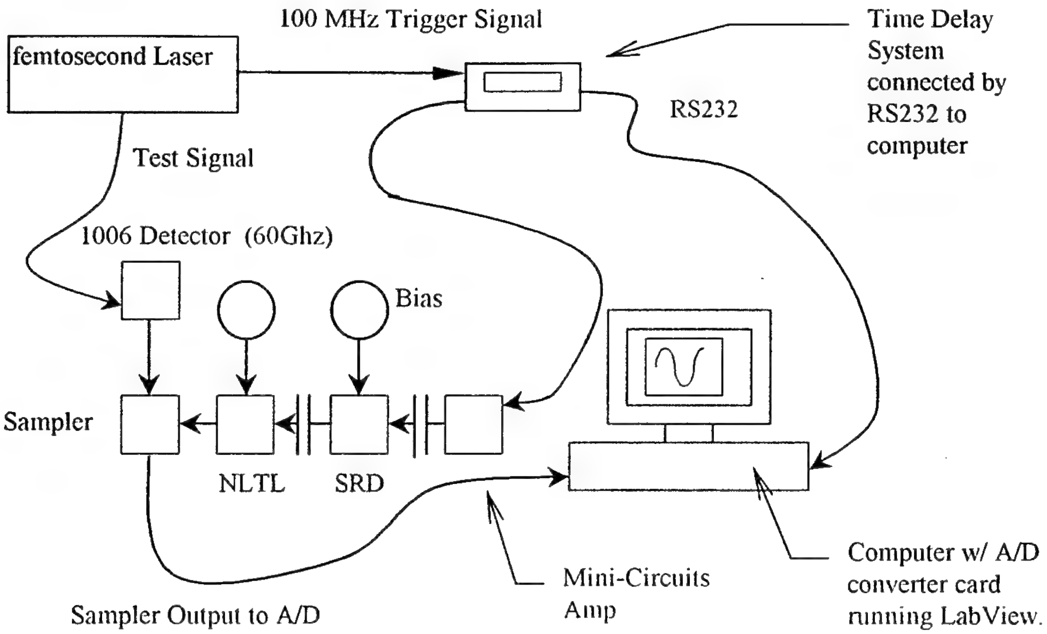


Figure 27: Prototype 2-ps optical sampling oscilloscope as described in text

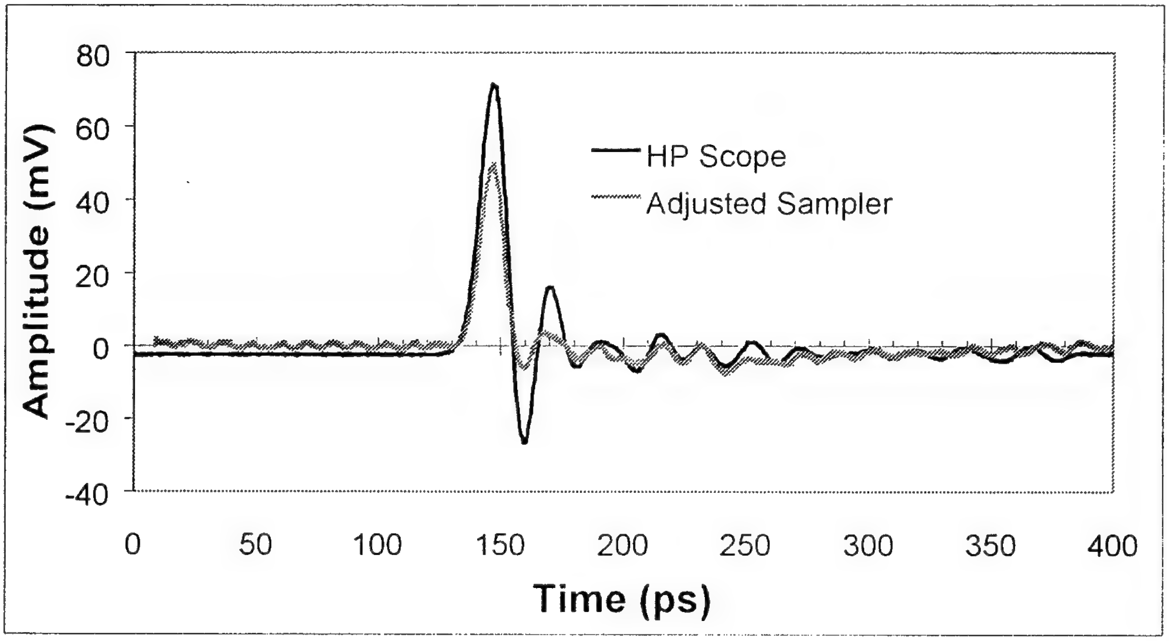


Figure 28: Comparison between sampling scope and HP54750

Final Integration

A Schottky diode consists of a Schottky contact to an N- active region and an ohmic contact to a N++ layer. Figure 29 shows a cross-sectional view of a Schottky diode structure.

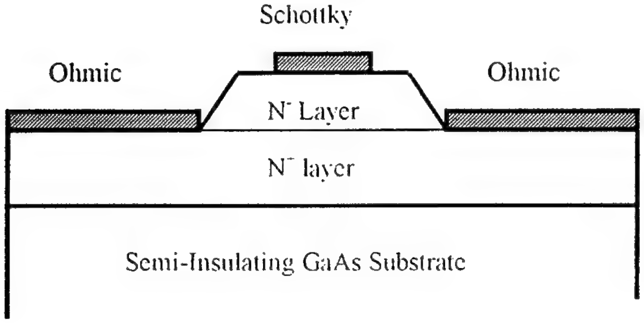


Figure 29: Cross Section of a planar Schottky diode.

In order to accommodate the Schottky diode structure along with the RTD structure, one more layer (a N^- layer) has to be added on top of the RTD structure. The N^- layer is used as the Schottky layer and the RTD's top N^{++} layer is used as the ohmic layer to produce a Schottky diode. Figure 30 shows the suggested structure. The active area of the diodes is defined by the intersection of the Schottky metal (same as interconnect metal) and the region protected form proton damage. Mask 1, which is used to make the RTDs' top contacts, contains the necessary pattern for the Schottky diodes' ohmic contact too.

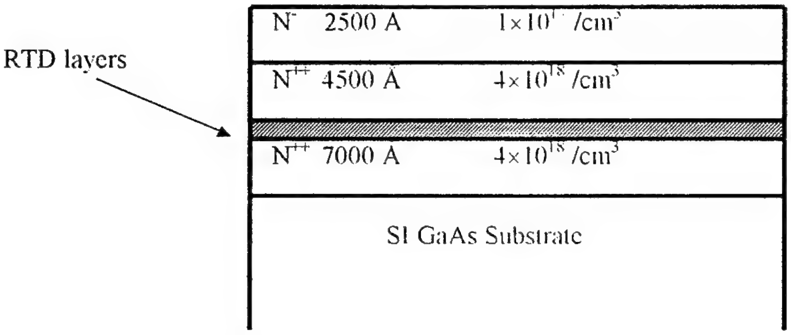

N^- 2500 A 1×10^{17} /cm^3
RTD layers
N^{++} 4500 Å 4×10^{18} /cm^3
N^{++} 7000 A 4×10^{18} /cm^3
SI GaAs Substrate

Figure 30: The layer structure suggested for the integration of RTDs and Schottky diodes.

Using the above process, Figure 31 shows the picture of a trigger circuit made on semi-insulating substrate. Currently, fabrication of sampling circuits based on integration of RTDs and Schottky diodes is in process.

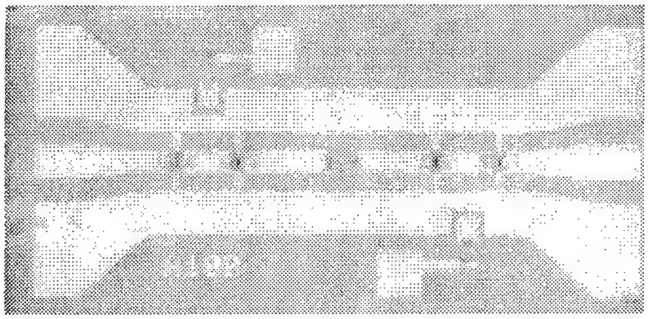


Figure 31: Picture of a trigger circuit made on semi-insulating substrate.

Appendix 1 – Command Set

```
:TIMEbase                Controls all horizontal sweep functions
        :REFerence[?] <ref_value>[suffix]        Sets the delay starting point.
                         <ref_value>             Time in seconds.
                         [suffix] ::= ns, ps, fs}        (Valid range: 0E0 to 4ns)
        :RANGe[?]        <full_scale_range>[suffix]      Sets the full-scale horizontal time.
                         <full_scale_range>      Time in seconds.
                         [suffix] ::= ns, ps, fs}        (Valid range: 10fs to 4ns)
        :RECordlength[?] <length>                        Sets record length for sweep.
                         <length>                        Number (Valid range: 1 to 10000)
        :SCALe[?]        <time_per_step>[suffix] Sets the delay step resolution.
                         <time_per_step> Time in seconds.
                         [suffix] ::= ns, ps, fs}        (Valid range: 10fs to 4ns)
:SOURce[?] {EXTernal | FREerun}                  Selects step trigger.
                         External is the hardware trigger.
:TRIGger                 Controls conditions for triggering.
        :DELay[?]        <delay_value>[suffix]           Sets the time between step
changes.
                         <delay_value>   Time in seconds.
                         [suffix] ::= us, ms, s}         (Valid range: 1us to 10s)
        :SLOPe[?] {POSitive | NEGative}          Specifies sweep trigger edge.
                                                         Valid only in external trigger
mode.
        :SOURce[?] {EXTernal | FREerun}                  Selects  sweep trigger.
                                                         External is the hardware trigger.
[:OUTPut]                Controls delay changes
        :STEP                                            Does a delay step.
        [:SWEEp]                                         Controls sweeps
                :SINGle                          Begins a single sweep.
                :RUN                                     Begins repetitive sweeps.
                :STOP                                    Stops sweeps.
```

```
:TIMEbase                Controls all horizontal sweep functions
        :REFerence[?] <ref_value>[suffix]        Sets the delay starting point.
                         <ref_value>             Time in seconds.
                         [suffix] ::= ns, ps, fs}        (Valid range: 0E0 to 4ns)
        :RANGe[?]        <full_scale_range>[suffix]      Sets the full-scale horizontal time.
                         <full_scale_range>      Time in seconds.
                         [suffix] ::= ns, ps, fs}        (Valid range: 10fs to 4ns)
        :RECordlength[?] <length>                        Sets record length for sweep.
                         <length>                        Number (Valid range: 1 to 10000)
        :SCALe[?]        <time_per_step>[suffix] Sets the delay step resolution.
                         <time_per_step> Time in seconds.
                         [suffix] ::= ns, ps, fs}        (Valid range: 10fs to 4ns)
:SOURce[?] {EXTernal | FREerun}                  Selects step trigger.
                         External is the hardware trigger.
:TRIGger                 Controls conditions for triggering.
        :DELay[?]        <delay_value>[suffix]           Sets the time between step
changes.
                         <delay_value>   Time in seconds.
                         [suffix] ::= us, ms, s}         (Valid range: 1us to 10s)
```

```
          :SLOPe[?] {POSitive | NEGative}      Specifies sweep trigger edge.
                                                         Valid only in external trigger
mode.
          :SOURce[?] {EXTernal | FREerun}                Selects  sweep trigger.
                                                         External is the hardware trigger.
[:OUTPut]                   Controls delay changes
          :STEP                                          Does a delay step.
          [:SWEEp]                                       Controls sweeps
                    :SINGle                    Begins a single sweep.
                    :RUN                                 Begins repetitive sweeps.
                    :STOP                                Stops sweeps.
```

Appendix 2 – Code Files

Firmware code files specifically modified from Picobox for AFOSR.

Delays.c

Remote.c

Remfuncs.c, Remfuncs.h

Globdef.h

Constdef.h

Remcmds.h

```
//
// AFOSR Command Definitions,  REMCMDS.H

#define   __ACQ              3630
#define   __ACQUIRE          31141
#define   __AUTO             51825
#define   __CAL              2477
#define   __CALIBRATE        43878
#define   __COAR             57587
#define   __COARSE           8641
#define   __COMM             43412
#define   __COMMAND          46970
#define   __DATA             4278
#define   __DEL              2538
#define   __DELAY            37890
#define   __EXT              4624
#define   __EXTERNAL         14437
#define   __FINE             16550
#define   __FS               275
#define   __INT              4478
#define   __INTERNAL         14291
#define   __NEG              1423
#define   __NEGATIVE         50763
#define   __NS               283
#define   __PLUG             24930
#define   __PLUGIN           12653
#define   __POIN             45900
#define   __POINTS           61997
#define   __POS              4275
#define   __POSITIVE         15079
#define   __PS               285
#define   __RANG             23192
#define   __RANGE            29084
#define   __REF              1202
#define   __REFERENCE        55178
#define   __RES              4127
#define   __RESET            61998
#define   __RUN              3242
#define   __SCAL             37173
#define   __SCALE            43065
#define   __SING             23313
#define   __SINGLE           13208
#define   __SLOP             53958
#define   __SLOPE            59850
#define   __SOUR             62103
#define   __SOURCE           55125
#define   __STEP             51828
#define   __STOP             54078
#define   __SWEE             14748
#define   __SWEEP            53227
#define   __TIME             16339
#define   __TIMEBASE         64162
#define   __TIMER            24996
#define   __TRIG             22324
#define   __TRIGGER          2983
#define   __US               290
```

```
//
// AFOSR command parser, REMOTE.C

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include <ds8xc520.h>
#include "remcmds.h"
#include "globref.h"
#include "constdef.h"
#include "remfuncs.h"


bit TakeAction() {

  xdata unsigned char callIndex;

  bit  takeActionReturn;

  switch(getToken()) {

    case __ATSIGN:
      ResetSerialCommand();
      break;
    case __CLS:
      CLS();
      break;
    case __ESE:
      switch(getToken()) {
        case __QUESTIONMARK:
          QueryESE();
          break;
        case __NUMBER:
          SetESE();
          break;
        default:
          COMError = TRUE;
          break;
      }
      break;
    case __ESR:
      switch(getToken()) {
        case __QUESTIONMARK:
          QueryESR();
          break;
        default: /*  error */
          COMError = TRUE;
          break;
      }
      break;
    case __IDN:
      switch(getToken()) {
        case __QUESTIONMARK:
          QueryIDN();
          break;
        default: /*  error */
          COMError = TRUE;
          break;
      }
      break;
    case __OPC:
      switch(getToken()) {
        case __QUESTIONMARK:
          QueryOPC();
          break;
        default:
          OPC();
          break;
      }
      break;
    case __OPT:
```

```
      switch(getToken()) {
        case __QUESTIONMARK:
          QueryOPT();
          break;
        default: /*  error */
          COMError = TRUE;
          break;
      }
      break;
    case __RST:
      RST();
      break;
    case __SRE:
      switch(getToken()) {
        case __QUESTIONMARK:
          QuerySRE();
          break;
        case __NUMBER:
          SetSRE();
          break;
        default:
          COMError = TRUE;
          break;
      }
      break;
    case __STB:
      switch(getToken()) {
        case __QUESTIONMARK:
          QuerySTB();
          break;
        default: /*  error */
          COMError = TRUE;
          break;
      }
      break;
    case __TST:
      switch(getToken()) {
        case __QUESTIONMARK:
          QueryTST();
          break;
        default: /*  error */
          COMError = TRUE;
          break;
      }
      break;
    case __WAI:
      WAI();
      break;
    case __GTL:
      GTL();
      break;
    case __HIDDEN:
      switch(getToken()) {
        case __LOADGPIBFLASH:
          LoadGPIBFlash();
          break;
        case __DEL:
        case __DELAY:
          switch(getToken()) {
            case __NUMBER:
              switch(unitsValue) {
                case __NONE:
// unitless, bounded by number of timesteps
                  QueryDelayValue(numberValue);
                    break;
                  default:
                    COMError = TRUE;
                    break;
                }
                break;
              default:
              COMError = TRUE;
          break;
        }
          break;
        case __RES:
          switch(getToken()) {
            case __NUMBER:
              switch(unitsValue) {
                case __NONE:
// unitless, bounded by number of timesteps
                  QueryResidual(numberValue);
                  break;
                default:
                  COMError = TRUE;
                  break;
              }
              break;
            default:
              COMError = TRUE;
              break;
          }
          break;
        default:
          COMError = TRUE;
          break;
      }
      break;


    // -----------------------------------------------------------
    //
    // AFOSR Command set
    //
    // -----------------------------------------------------------


    case __TIME    :
    case __TIMEBASE :
      switch(getToken()) {
        case __DEL:
        case __DELAY:
          switch(getToken()) {
            case __NUMBER:
              switch(unitsValue) {
                case __NONE:  // assume uS
                  StepDelaySet(numberValue);
                  break;
                case __US:
                  StepDelaySet(numberValue);
                  break;
                case __MS:
                  StepDelaySet(numberValue * 1000);
                  break;
                case __S:
                  StepDelaySet(numberValue * 1000000);
                  break;
                default:
                  COMError = TRUE;
                  break;
              }
              break;
            case __QUESTIONMARK:
              StepDelayQuery();
              break;
            default:
              COMError = TRUE;
              break;
          }
          break;
        case __SLOP:
        case __SLOPE:
          switch(getToken()) {
            case __POS:
            case __POSITIVE:
              StepSlopeSet(POSITIVE);
              break;
            case __NEG:
            case __NEGATIVE:
              StepSlopeSet(NEGATIVE);
              break;
            case __QUESTIONMARK:
              StepSlopeQuery();
              break;
            default:
              COMError = TRUE;
              break;
          }
          break;
        case __SOUR:
        case __SOURCE:
          switch(getToken()) {
            case __EXT:
            case __EXTERNAL:
              StepSourceSet(EXTERNAL);
              break;
            case __INT:
            case __INTERNAL:
              StepSourceSet(INTERNAL);
              break;
            case __QUESTIONMARK:
              StepSourceQuery();
              break;
            default:
              COMError = TRUE;
              break;
          }
          break;
        case __REF:
        case __REFERENCE:
          switch(getToken()) {
            case __NUMBER:
              switch(unitsValue) {
                case __NONE:  // assume fS
                  TimebaseReferenceSet(numberValue);
                  break;
                case __FS:
                  TimebaseReferenceSet(numberValue);
                  break;
                case __PS:
                  TimebaseReferenceSet(numberValue * 1000);
                  break;
                case __NS:
                  TimebaseReferenceSet(numberValue * 1000000);
                  break;
                default:
                  COMError = TRUE;
                  break;
              }
              break;
            case __QUESTIONMARK:
              TimebaseReferenceQuery();
              break;
            default:
              COMError = TRUE;
              break;
          }
          break;
        case __RANG:
        case __RANGE:
          switch(getToken()) {
            case __NUMBER:
              switch(unitsValue) {
                case __NONE:  // assume fS
                  TimebaseRangeSet(numberValue);
                  break;
                case __FS:
                  TimebaseRangeSet(numberValue);
              break;
            case __PS:
              TimebaseRangeSet(numberValue * 1000);
              break;
            case __NS:
              TimebaseRangeSet(numberValue * 1000000);
              break;
            default:
              COMError = TRUE;
              break;
          }
          break;
        case __QUESTIONMARK:
          TimebaseRangeQuery();
          break;
        default:
          COMError = TRUE;
          break;
      }
      break;
    case __REC:
    case __RECORDLENGTH:
      switch(getToken()) {
        case __NUMBER:
          switch(unitsValue) {
            case __NONE:  // unitless - bounded 1 to 10000
              TimebaseRecLenSet((unsigned int) numberValue);
              break;
            default:
              COMError = TRUE;
              break;
          }
          break;
        case __QUESTIONMARK:
          TimebaseRecLenQuery();
          break;
        default:
          COMError = TRUE;
          break;
      }
      break;
    case __SCAL:
    case __SCALE:
      switch(getToken()) {
        case __NUMBER:
          switch(unitsValue) {
            case __NONE:  // assume fS
              TimebaseScaleSet(numberValue);
              break;
            case __FS:
              TimebaseScaleSet(numberValue);
              break;
            case __PS:
              TimebaseScaleSet(numberValue * 1000);
              break;
            case __NS:
              TimebaseScaleSet(numberValue * 1000000);
              break;
            default:
              COMError = TRUE;
              break;
          }
          break;
        case __QUESTIONMARK:
          TimebaseScaleQuery();
          break;
        default:
          COMError = TRUE;
          break;
      }
      break;
    default:
      COMError = TRUE;
      break;
  }
  break;


case __TRIG   :
case __TRIGGER:
  switch(getToken()) {
    case __DEL:
    case __DELAY:
      switch(getToken()) {
        case __NUMBER:
          switch(unitsValue) {
            case __NONE:  // assume uS
              SweepDelaySet(numberValue);
              break;
            case __US:
              SweepDelaySet(numberValue);
              break;
            case __MS:
              SweepDelaySet(numberValue * 1000);
              break;
            case __S:
              SweepDelaySet(numberValue * 1000000);
              break;
            default:
              COMError = TRUE;
              break;
          }
          break;
        case __QUESTIONMARK:
          SweepDelayQuery();
          break;
        default:
          COMError = TRUE;
          break;
      }
      break;
    case __SLOP:
    case __SLOPE:
      switch(getToken()) {
        case __POS:
        case __POSITIVE:
          SweepSlopeSet(POSITIVE);
          break;
        case __NEG:
        case __NEGATIVE:
          SweepSlopeSet(NEGATIVE);
          break;
        case __QUESTIONMARK:
          SweepSlopeQuery();
          break;
        default:
          COMError = TRUE;
          break;
      }
      break;
    case __SOUR:
    case __SOURCE:
      switch(getToken()) {
        case __EXT:
        case __EXTERNAL:
          SweepSourceSet(EXTERNAL);
          break;
        case __INT:
        case __INTERNAL:
          SweepSourceSet(INTERNAL);
          break;
        case __QUESTIONMARK:
          SweepSourceQuery();
          break;
        default:
          COMError = TRUE;
          break;
      }
      break;
    default:
      COMError = TRUE;
      break;
  }
  break;

case __ACQ:
case __ACQUIRE:
  switch(getToken()) {
    case __POIN:
    case __POINTS:
      switch(getToken()) {
        case __NUMBER:
          AcquirePoints(numberValue);
          break;
        case __AUTO:
          AcquirePoints(0);
          break;
        case __QUESTIONMARK:
          AcquirePointsQuery();
          break;
        default:
          COMError = TRUE;
          break;
      }
      break;
    default:
      COMError = TRUE;
      break;
  }
  break;

case __OUTP:
case __OUTPUT:
  switch(getToken()) {
    case __STEP:
      switch(getToken()) {
        case __NONE:  // assume 1 step
          OutputStep(1);
          break;
        case __NUMBER:
          OutputStep((unsigned int) numberValue);
          break;
        case __QUESTIONMARK:
          OutputStepQuery();
          break;
        default:
          OutputStep(1);
          break;
      }
      break;
    case __SWEE:
    case __SWEEP:
      switch(getToken()) {
        case __SING:
        case __SINGLE:
          OutputSweep(SINGLE);
          break;
        case __RUN:
          OutputSweep(RUN);
          break;
        case __STOP:
          OutputSweepStop();
          break;
        default:
          COMError = TRUE;
          break;
      }
      break;
    default:
      COMError = TRUE;
      break;
  }
  break;


//
// Default case of [:OUTPut]:STEP
//

case __STEP:
  switch(getToken()) {
    case __NONE:  // assume 1 step
      OutputStep(1);
      break;
    case __NUMBER:
      OutputStep((unsigned int) numberValue);
      break;
    case __QUESTIONMARK:
      OutputStepQuery();
      break;
    default:
      OutputStep(1);
  }
  break;

//
// Default case of [:OUTPut][:SWEEp]
//

case __SWEE:
case __SWEEP:
  switch(getToken()) {
    case __SING:
    case __SINGLE:
      OutputSweep(SINGLE);
      break;
    case __RUN:
      OutputSweep(RUN);
      break;
    case __STOP:
      OutputSweepStop();
      break;
    default:
      COMError = TRUE;
      break;
  }
  break;

//
// Default case of [:OUTPut][:SWEEp]:SINGle
//

case __SING:
case __SINGLE:
  OutputSweep(SINGLE);
  break;

//
// Default case of [:OUTPut][:SWEEp]:RUN
//

case __RUN:
  OutputSweep(RUN);
  break;

//
// Default case of [:OUTPut][:SWEEp]:STOP
//
case __STOP:
  OutputSweepStop();
  break;
```

```
    case __COAR:
    case __COARSE:
      switch(getToken()) {
        case __NUMBER:
          CoarseDelaySet(numberValue);
          break;
        case __QUESTIONMARK:
          CoarseValueQuery();
          break;
        default:
          COMError = TRUE;
          break;
      }
      break;

    case __FINE:
      switch(getToken()) {
        case __NUMBER:
          FineDelaySet(numberValue);
          break;
        case __QUESTIONMARK:
          FineValueQuery();
          break;
        default:
          COMError = TRUE;
          break;
      }
      break;


    case __CAL:
    case __CALIBRATE:
      switch(getToken()) {
        case __PLUG:
        case __PLUGIN:
          switch(getToken()) {
            case __DATA:
              switch(getToken()) {
                case __COAR:
                case __COARSE:
                  switch(getToken()) {
                    case __NUMBER:
                      callIndex = (unsigned char) numberValue;
                      switch(getToken()) {
                        case __COMMA:
                          switch(getToken()) {
                            case __NUMBER:
                              CalibratePlugin(COARSE, callIndex,
                                longNumberValue, 0);
                              // need to bring slotnum and data in
                              break;
                            default:
                              COMError = TRUE;
                              break;
                          }
                          break;
                        default:
                          COMError = TRUE;
                          break;
                      }
                      break;
                    case __QUESTIONMARK:
                      QueryCalibratePluginChecksum(COARSE);
                      break;
                    default:
                      COMError = TRUE;
                      break;
                  }
                  break;
                case __FINE:
                  switch(getToken()) {
                    case __NUMBER:
                      callIndex = (unsigned char) numberValue;
                      switch(getToken()) {
                        case __COMMA:
                          switch(getToken()) {
                            case __NUMBER:
                              CalibratePlugin(FINE, callIndex, 0,
                                numberValue);
                              // need to bring slotnum and data in
                              break;
                            default:
                              COMError = TRUE;
                              break;
                          }
                          break;
                        default:
                          COMError = TRUE;
                          break;
                      }
                      break;
                    case __QUESTIONMARK:
                      QueryCalibratePluginChecksum(FINE);
                      break;
                    default:
                      COMError = TRUE;
                      break;
                  }
                  break;
                default:
                  COMError = TRUE;
                  break;
              }
              break;
            default:
              COMError = TRUE;
              break;
          }
          break;
        default:
          COMError = TRUE;
          break;
      }
      break;

    default:    /*  error */
      COMError = TRUE;
      break;
  } /* first switch */


  /* flag command error */
  if (COMError == TRUE) {
    sprintf(outbuf,"Unknown Command");
    COMError = FALSE;
  }

  // check for additional commands on this command line

  switch(getToken()) {
    case __ENDOFCOMMAND:
      takeActionReturn = _MORECOMMANDS;       // 1
      break;
    default:
      takeActionReturn = _NOMORECOMMANDS;     // 0
      break;
  }

  return takeActionReturn;


} // TakeAction

// REMOTE.C
```

```
//
// AFOSR Function Prototypes, REMFUNCS.H

extern void  TimebaseReferenceSet(float);
extern void  TimebaseReferenceQuery(void);
extern void  TimebaseRangeSet(float);
extern void  TimebaseRangeQuery(void);
extern void  TimebaseRecLenSet(unsigned int);
extern void  TimebaseRecLenQuery(void);
extern void  TimebaseScaleSet(float);
extern void  TimebaseScaleQuery(void);

extern void  SweepDelaySet(float);
extern void  SweepDelayQuery(void);
extern void  SweepSlopeSet(enum type_Slope Slope);
extern void  SweepSlopeQuery(void);
extern void  SweepSourceSet(enum type_TriggerSource Source);
extern void  SweepSourceQuery(void);

extern void  StepDelaySet(float);
extern void  StepDelayQuery(void);
extern void  StepSlopeSet(enum type_Slope Slope);
extern void  StepSlopeQuery(void);
extern void  StepSourceSet(enum type_TriggerSource Source);
extern void  StepSourceQuery(void);

extern void  AcquirePoints(unsigned long);
extern void  AcquirePointsQuery(void);

extern void  OutputStep(unsigned int);
extern void  OutputStepQuery(void);
extern void  OutputSweep(enum type_SweepType);
extern void  OutputSweepStop(void);

extern void  CalibratePlugin(enum type_CalData CalDataType, unsigned char index, unsigned long lvalue, float fvalue); // need to bring
slotnum and data in
extern void  QueryCalibratePluginChecksum(enum type_CalData CalDataType);

extern void QueryDelayValue(unsigned int);
extern void QueryResidual(unsigned int);

extern void  CoarseDelaySet(float);
extern void  FineDelaySet(float);

extern void  CoarseValueQuery(void);
extern void  FineValueQuery(void);
```

```
// AFOSR functions, REMFUNCS.C
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include "absolute.h"
#include "globref.h"
#include "constdef.h"
#include "remfuncs.h"

// External Function Declarations
extern void updateDisplay(void);
extern unsigned int CalculatedFineDelayNumber(unsigned long,
unsigned long *);
extern unsigned int CalculatedCoarseDelayNumber(unsigned long ,
unsigned long *);
extern void  SetDelay(unsigned int);
extern void  ToggleAtoDStrobe(void);
extern void  StartTimer(unsigned long);
extern void  EnableSweepTriggerInterrupt(void);

void ResetSerialCommand(void) {

  sprintf(outbuf,"#");
}
void CLS(void) {

  sprintf(outbuf,"CLS Not Implemented");
}
void QueryESE(void) {

  sprintf(outbuf,"Query ESE Not Implemented");
}
void SetESE(void) {

  sprintf(outbuf,"Set ESE Not Implemented");
}
void QueryESR(void) {

  sprintf(outbuf,"Query ESR Not Implemented");
}
void QueryIDN(void) {

  sprintf(outbuf,"FocusedResearch DigitalDelay
G%4.2f",((float)FIRMWAREVERSION)/100);
}

void QueryOPC(void) {

  if (sweeping) {
    sprintf(outbuf,"0");
  }
  else {
    sprintf(outbuf,"1");
  }
}

void OPC(void) {

  sprintf(outbuf,"Not Implemented");
}

void QueryOPT(void) {

  sprintf(outbuf,"No Options");
}

void RST(void) {

  DisplayControlMask &= 0xEF; /* Turn on Remote LED */
  DisplayControl = DisplayControlMask;
  CurrentState = REMOTE;

  sprintf(outbuf, "OK");
}

void QuerySRE(void) {

  sprintf(outbuf,"Query SRE Not Implemented");
}

void SetSRE(void) {

  sprintf(outbuf,"Set SRE Not Implemented");
}

void QuerySTB(void) {

  sprintf(outbuf,"Query STB Not Implemented");
}

void QueryTST(void) {

  // returns a self-test successfully completed message.
  // value returned should be between -32767 and 32767
  // with zero being a successful test.
  sprintf(outbuf,"0");
}

void WAI(void) {

  sprintf(outbuf,"WAI Not Implemented");
}

void GTL(void) {

  DisplayControlMask |= 0x10;     /* Turn off Remote LED */
  DisplayControl = DisplayControlMask;
  NextState = LOCAL;
  sprintf(outbuf,"OK");
}

void LoadGPIBFlash(void) {

  sprintf(dispbuf,"  Download....  ");
  updateDisplay();  // notify world that we're doing something
  sprintf(outbuf,"Downloading GPIB FLASH");

  // sets ROMSIZE to 16K and resets processor to
  // begin execution from internal PROM
  //
  enterBootLoader();
}
```

```
// ----------------------------------------------------------
// AFOSR Commands
// ----------------------------------------------------------

void TimebaseReferenceSet(float Reference) {

  if ((Reference >= MIN_TB_REF) &&
    (Reference <= MAX_TB_REF) ) {
    timebaseReference = (unsigned long) Reference*UNITSCALE;
    timebaseCurrentStep = 0;
    delayValuesNeedToBeRecalculated = TRUE;
    sprintf(outbuf, "OK");
  }
  else { // out of range
    /* no action */
    sprintf(outbuf, "Out of Range");
  }
}

void TimebaseReferenceQuery(void) {

  sprintf(outbuf,"%8.2G",
      ( (float) timebaseReference/UNITSCALE) );
}

void TimebaseRangeSet(float Range) {

  if ((Range >= MIN_TB_RANGE) && (Range <= MAX_TB_RANGE)
) {
    timebaseRange = (unsigned long) Range*UNITSCALE;
    timebaseNumberOfSteps =
MIN(ceil(timebaseRange/timebaseScale), MAXNUMSTEPS);
    timebaseCurrentStep = 0;
    delayValuesNeedToBeRecalculated = TRUE;
    sprintf(outbuf, "OK");
  }
  else { // out of range
    /* no action */
    sprintf(outbuf, "Out of Range");
  }

}

void TimebaseRangeQuery(void){

  sprintf(outbuf,"%8.2G",((float) timebaseRange/UNITSCALE));
}

void TimebaseRecLenSet(unsigned int RecLen) {

  if ((RecLen >= 1) && (RecLen <= MAXNUMSTEPS))  {
    timebaseScale = MIN(ceil(timebaseRange/RecLen),
MAX_TB_SCALE);
    timebaseNumberOfSteps =
MIN(ceil(timebaseRange/timebaseScale), RecLen);

    timebaseCurrentStep = 0;
    delayValuesNeedToBeRecalculated = TRUE;
    sprintf(outbuf, "OK");
  }
  else { // out of range
    // no action
    sprintf(outbuf, "Out of Range");
  }
}

void TimebaseRecLenQuery(void) {

  sprintf(outbuf,"%u", timebaseNumberOfSteps);
}

void TimebaseScaleSet(float Scale) {

  if ((Scale >= MIN_TB_SCALE) && (Scale <= MAX_TB_SCALE) ) {
    timebaseScale = (unsigned long) Scale*UNITSCALE;
    timebaseNumberOfSteps =
MIN(ceil(timebaseRange/timebaseScale), MAXNUMSTEPS);
    timebaseCurrentStep = 0;
    delayValuesNeedToBeRecalculated = TRUE;
    sprintf(outbuf, "OK");
  }
  else { // out of range
    /* no action */
    sprintf(outbuf, "Out of Range");
  }
}

void TimebaseScaleQuery(void) {

  sprintf(outbuf,"%8.2G",((float) timebaseScale/UNITSCALE));
}

void StepDelaySet(float Delay) {

  if ((Delay >= MIN_DELAY) && (Delay <= MAX_DELAY) ) {
    stepTriggerDelay = (unsigned long) Delay;
    sprintf(outbuf, "OK");
  }
  else { // out of range
    /* no action */
    sprintf(outbuf, "Out of Range");
  }
}

void StepDelayQuery(void) {

  sprintf(outbuf,"%8.2G",( (float) stepTriggerDelay) );
}

void SweepDelaySet(float Delay) {

  if ((Delay >= MIN_DELAY) && (Delay <= MAX_DELAY) ) {
    sweepTriggerDelay = (unsigned long) Delay;
    sprintf(outbuf, "OK");
  }
  else { // out of range
    /* no action */
    sprintf(outbuf, "Out of Range");
  }
}

void SweepDelayQuery(void) {

  sprintf(outbuf,"%8.2G",(float) sweepTriggerDelay);
}

void StepSlopeSet(enum type_Slope Slope) {

  switch(Slope) {
    case NEGATIVE:
      stepTriggerSlope = NEGATIVE;
      sprintf(outbuf, "OK");
      break;
    case POSITIVE:
      stepTriggerSlope = POSITIVE;
      sprintf(outbuf, "OK");
      break;
    default:  // out of range
      /* no action */
      sprintf(outbuf, "Out of Range");
      break;
  }
}

void StepSlopeQuery(void) {
  switch(stepTriggerSlope) {
    case NEGATIVE:
      sprintf(outbuf, "NEGATIVE");  // negative slope
      break;
    case POSITIVE:
      sprintf(outbuf, "POSITIVE"); // positive slope
      break;
    default:
      sprintf(outbuf, "Unknown Slope");
      break;
  }
}


void SweepSlopeSet(enum type_Slope Slope) {

  switch(Slope) {
    case NEGATIVE:
      sweepTriggerSlope = NEGATIVE;
      sprintf(outbuf, "OK");
      break;
    case POSITIVE:
      sweepTriggerSlope = POSITIVE;
      sprintf(outbuf, "OK");
      break;
    default:  // out of range
      /* no action */
      sprintf(outbuf, "Out of Range");
      break;
  }
}

void SweepSlopeQuery(void) {
  switch(sweepTriggerSlope) {
    case NEGATIVE:
      sprintf(outbuf, "NEGATIVE");  // negative slope
      break;
    case POSITIVE:
      sprintf(outbuf, "POSITIVE"); // positive slope
      break;
    default:
      sprintf(outbuf, "Unknown Slope");
      break;
  }
}

void StepSourceSet(enum type_TriggerSource Source) {
  switch (Source) {
    case INTERNAL:
      stepTriggerSource = INTERNAL;
      sprintf(outbuf, "OK");
      break;
    case EXTERNAL:
      stepTriggerSource = EXTERNAL;
      sprintf(outbuf, "OK");
      break;
```

```
    default:
      // out of range
      /* no action */
      sprintf(outbuf, "Out of Range");
      break;
  }
}

void StepSourceQuery(void) {
  switch(stepTriggerSource) {
    case INTERNAL:
      sprintf(outbuf, "INTERNAL");  // Timed
      break;
    case EXTERNAL:
      sprintf(outbuf, "EXTERNAL");  // External
      break;
    default:
      sprintf(outbuf, "Unknown Cardtype");
      break;
  }
}


void SweepSourceSet(enum type_TriggerSource Source) {

  switch (Source) {
    case INTERNAL:
      sweepTriggerSource = INTERNAL;
      sprintf(outbuf, "OK");
      break;
    case EXTERNAL:
      sweepTriggerSource = EXTERNAL;
      sprintf(outbuf, "OK");
      break;
    default:
      // out of range
      /* no action */
      sprintf(outbuf, "Out of Range");
      break;
  }
}

void SweepSourceQuery(void) {
  switch(sweepTriggerSource) {
    case INTERNAL:
      sprintf(outbuf, "INTERNAL");  // Timed
      break;
    case EXTERNAL:
      sprintf(outbuf, "EXTERNAL");  // External
      break;
    default:
      sprintf(outbuf, "Unknown Source");
      break;
  }

}

void OutputStep(unsigned int numberSteps) {

  xdata unsigned long coarseResidual;
  xdata unsigned long fineResidual;
  xdata unsigned int i;

  if (delayValuesNeedToBeRecalculated) {
    // Fill array with delay values to use
    // - faster during realtime operations
    //
    for (i=0; i<timebaseNumberOfSteps; i++) {

      coarseDelayValue[i] =
CalculatedCoarseDelayNumber(timebaseReference + (i *
timebaseScale), &coarseResidual);
      fineDelayValue[i] =
CalculatedFineDelayNumber(coarseResidual, &fineResidual);
      residual[i] = fineResidual;
    }
    delayValuesNeedToBeRecalculated = FALSE;
  }

  for (i=0; i<numberSteps; i++) {

    // steps the current delay by one scale size.
    if (timebaseCurrentStep >= timebaseNumberOfSteps) {
    // roll over step if at end of travel
      timebaseCurrentStep = 0;
    }

    SetDelay(timebaseCurrentStep++);
    ToggleAtoDStrobe();       // use EXT0 interrupt pin P3.2.
  }
  sprintf(outbuf, "OK");
}

void OutputStepQuery(void) {

  sprintf(outbuf,"%u", timebaseCurrentStep);
}


void OutputSweep(enum type_SweepType Type) {

  data unsigned long i;
  xdata unsigned long coarseResidual;
  xdata unsigned long fineResidual;

  if (delayValuesNeedToBeRecalculated) {
    // Fill array with delay values to use
    // - faster during realtime operations
    for (i=0; i<timebaseNumberOfSteps; i++) {
      coarseDelayValue[i] =
CalculatedCoarseDelayNumber(timebaseReference + (i *
timebaseScale), &coarseResidual);
      fineDelayValue[i] =
CalculatedFineDelayNumber(coarseResidual, &fineResidual);
      residual[i] = fineResidual;
    }
    delayValuesNeedToBeRecalculated = FALSE;
  }

  switch (Type) {
    case (SINGLE):
      sweepType = SINGLE;
      break;
    case (RUN):
      sweepType = RUN;
      break;
    default:
      sweepType = SINGLE;
      break;
  }

    switch(sweepTriggerSource) {
      case INTERNAL:
      sweeping = TRUE;
```

```
      timebaseCurrentStep = 0;
      StartTimer(sweepTriggerDelay);
      break;
    case EXTERNAL:
      sweeping = FALSE;
      EnableSweepTriggerInterrupt();
      break;
    default:
      sweepTriggerSource = INTERNAL;
      sweeping = TRUE;
      timebaseCurrentStep = 0;
      StartTimer(sweepTriggerDelay);
      break;
  }
  sprintf(outbuf, "OK");
}


void OutputSweepStop(void) {

  TR0 = 0;    // disable Timer 0    //TCON.4 = 0;
  ET0 = 0;    // disable Timer 0 interrupt
  EX1 = 0;    // Disable interrupt  //IE.2 = 0;
  EX2 = 0;    // Disable interrupt  //EIE.0 = 0;
  EX3= 0;     // Disable interrupt  //EIE.1 = 0;
  EX4 = 0;    // Disable interrupt  //EIE.2 = 0;

  sweeping = FALSE;
  sweepType = STOP;
  sprintf(outbuf, "OK");
}


void CalibratePlugin(enum type_CalData calDataType, unsigned
char index, unsigned long lvalue, float fvalue) {

  data union unionUnsignedLong coarseValue;
  data union unionFloat fineValue;
  char xdata * xdata charAddress;
  xdata unsigned char dataByte;

  switch (calDataType) {
    case COARSE:
      if ( (index >= MIN_CINDEX) && (index <= MAX_CINDEX) &&
        (lvalue >= MIN_TB_REF*UNITSCALE) && (lvalue <=
MAX_TB_REF*UNITSCALE) ) {

        coarseValue.all = lvalue;
        charAddress = (char xdata *)
&NV_coarseDelayArray[index];

        dataByte = coarseValue.byte.b3;
        outp(charAddress, dataByte);
        while (inp(charAddress) != dataByte) {
        }
        charAddress++;
        dataByte = coarseValue.byte.b2;
        outp(charAddress, dataByte);
        while (inp(charAddress) != dataByte) {
        }

        charAddress++;
        dataByte = coarseValue.byte.b1;
        outp(charAddress, dataByte);
        while (inp(charAddress) != dataByte) {
        }
        charAddress++;
        dataByte = coarseValue.byte.b0;
        outp(charAddress, dataByte);
        while (inp(charAddress) != dataByte) {
        }
        delayValuesNeedToBeRecalculated = TRUE;
          // changed things
        sprintf(outbuf, "Ok");
      }
      else {
        sprintf(outbuf, "Coarse Out of Range");
      }
      break;

    case FINE:
      if ( (index >= MIN_FINDEX) && (index <= MAX_FINDEX) ) {
        fineValue.all = fvalue;
        charAddress = (char xdata *) &NV_fineCalCoeff[index];

        dataByte = fineValue.byte.b3;
        outp(charAddress, dataByte);
        while (inp(charAddress) != dataByte) {
        }
        charAddress++;
        dataByte = fineValue.byte.b2;
        outp(charAddress, dataByte);
        while (inp(charAddress) != dataByte) {
        }
        charAddress++;
        dataByte = fineValue.byte.b1;
        outp(charAddress, dataByte);
        while (inp(charAddress) != dataByte) {
        }
        charAddress++;
        dataByte = fineValue.byte.b0;
        outp(charAddress, dataByte);
        while (inp(charAddress) != dataByte) {
        }
        delayValuesNeedToBeRecalculated = TRUE;
          // changed things
        sprintf(outbuf, "Ok");
      }
      else {
        sprintf(outbuf, "Fine Out of Range");
      }
      break;
    default:
      sprintf(outbuf, "Unknown CalType");
      break;
  }
}

void QueryCalibratePluginChecksum(enum type_CalData
calDataType) {

  xdata unsigned long checksum;
  xdata float floatSum;
  xdata unsigned int i;

  checksum = 0;
  floatSum = 0;

  switch (calDataType) {
    case COARSE:
      for (i=MIN_CINDEX; i<=MAX_CINDEX; i++) {
        checksum += NV_coarseDelayArray[i];
      }
      break;
    case FINE:
      for (i=MIN_FINDEX; i<=MAX_FINDEX; i++) {
        floatSum += NV_fineCalCoeff[i];
      }
      checksum = (unsigned long) floor(floatSum);
      break;
  }
  sprintf(outbuf,"%lu",checksum);
}

void QueryDelayValue(unsigned int StepNumber) {

  sprintf(outbuf,"Step: %u\tCoarse: %u\tFine:
%u",StepNumber,coarseDelayValue[StepNumber],fineDelayValue[S
tepNumber]);
}

void QueryResidual(unsigned int StepNumber) {

  sprintf(outbuf,"Step: %u\tResidual: %u", StepNumber,
residual[StepNumber]);
}

void CoarseDelaySet(float delayValue) {

  if ((delayValue >= 0) && (delayValue <= NUM_COARSE) ) {

    coarseValueImage = (unsigned char) delayValue;
    coarseDelayRegister = (unsigned char) delayValue;   // C000
    coarseDelayStrobe = 1;                        // C001
    sprintf(outbuf, "OK");
  }
  else { // out of range
    /* no action */
    sprintf(outbuf, "Out of Range");
  }
}

void FineDelaySet(float delayValue) {

  data unsigned int tempDelay;

  if ((delayValue >= 0) && (delayValue <= NUM_FINE) ) {
    tempDelay = (unsigned int) delayValue;
    fineValueImage = tempDelay;
    fineDelayRegisterHigh =  (unsigned char) (tempDelay / 256);
  // C003 shift right by 8 bits
    fineDelayRegisterLow =    (unsigned char) (tempDelay &
0x00FF);  // C002 use only low 8 bits
    fineDelayStrobe = 1;        // C004 is this a pulse or level?
    sprintf(outbuf, "OK");
  }
  else { // out of range
    /* no action */
    sprintf(outbuf, "Out of Range");
  }
}

void CoarseValueQuery(void) {

  sprintf(outbuf, "%u",(unsigned int) coarseValueImage);
}

void FineValueQuery(void) {
  sprintf(outbuf, "%u",fineValueImage);
}
```

```
//
// AFOSR constant definitions,  CONSTDEF.H

#define MIN(A,B) ((A) < (B) ? (A) : (B))
#define MAX(A,B) ((A) > (B) ? (A) : (B))

#define UNITSCALE           10              // 0.1fs numbers
#define MAXNUMSTEPS         4096

#define MIN_CINDEX          0
#define MAX_CINDEX          255
#define NUM_COARSE          255             // 2^^8 coarse settings
#define MIN_FINDEX          0               // 3 for fine coefficients
#define MAX_FINDEX          3               // 1 for fine data (max value)
#define NUM_FINE            4095            // 2^^12 fine settings

#define DEF_TB_SCALE        1000            // 1ps
#define MIN_TB_REF          0               // 0
#define MAX_TB_REF          5000000         // 4ns
#define MIN_TB_RANGE        10              // 10fs
#define MAX_TB_RANGE        5000000         // 4ns
#define   MIN_TB_SCALE      10              // 10fs
#define MAX_TB_SCALE        5000000         // 4ns
#define MIN_DELAY           1               // 1us
#define MAX_DELAY           10000000        // 10sec

#define CLOCKFREQ           22.1184         // 22.1184 or 30 (in MHz)
#define DIVIDER             4               // 4 or 12
```

```
//
// AFOSR delay firmware, DELAYS.C

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include "constdef.h"
#include "globref.h"

// Finds the best coarse delay number given a desired Delay.
// Returns also the residual delay to be taken up by the fine delay.

unsigned int CalculatedCoarseDelayNumber(unsigned long desiredDelay, unsigned long * coarseResidual) {

// Assumes unsigned long coarseDelayArray[i] array of coarse delay calibration values
// Also unsigned long coarseResidual - passed on to fine delay.
// Look for closest coarse value then add more (around 20ps per step)

  xdata unsigned int bestCoarseNumber;
  data unsigned int i;

  bestCoarseNumber = MAX_CINDEX;          // Number to be sent to CoarseDelayGenerator on default
  for (i=0; i<=MAX_CINDEX; i++) {
    if (desiredDelay > NV_coarseDelayArray[i]) {
      continue;                           // keep looking
    }
    else {
      bestCoarseNumber = MAX(i-1,0);      // takes care of zero case and above (the one before)
      break;                              // to send to the coarse delay generaor
    }
  }

  *coarseResidual = desiredDelay - NV_coarseDelayArray[bestCoarseNumber];

  return (bestCoarseNumber);
}


unsigned int CalculatedFineDelayNumber(unsigned long desiredDelay, unsigned long * fineResidual) {

// Assumes global fineCalCoeff[0,1,2] coefficients for equation of line
// estimate of fine delays.

  xdata float fineDelayNumberEstimate;
  xdata float num, lowEstimate, highEstimate;
  xdata unsigned int fineDelayNumber;
  xdata unsigned long clippedDelay;

  clippedDelay = MIN(desiredDelay, NV_fineCalCoeff[3]);  // so delay does not go out of max range

  // quadratic coefficients
  num = sqrt((NV_fineCalCoeff[1]*NV_fineCalCoeff[1]) - (4.0 * NV_fineCalCoeff[2] * (NV_fineCalCoeff[0] - (float) clippedDelay)));
  fineDelayNumberEstimate = (-1.0 * NV_fineCalCoeff[1] - num) / (2.0 * NV_fineCalCoeff[2]);

  lowEstimate = floor(fineDelayNumberEstimate);          // move to the closest integer step
  highEstimate = ceil(fineDelayNumberEstimate);
  if ((highEstimate-fineDelayNumberEstimate) >= (fineDelayNumberEstimate-lowEstimate)) {
    fineDelayNumber = (unsigned int) lowEstimate;
  }
  else {
    fineDelayNumber = (unsigned int) highEstimate;
  }
  *fineResidual = desiredDelay -
      (unsigned long) (NV_fineCalCoeff[0]
      + (NV_fineCalCoeff[1] * (float) fineDelayNumber)
      + (NV_fineCalCoeff[2] * (float) (fineDelayNumber*fineDelayNumber)));

  return(fineDelayNumber);

}

void SetDelay(unsigned int i) {

    // CoarseDelaySet
    coarseDelayRegister = coarseDelayValue[i];        // C000
    coarseDelayStrobe = 1;                            // C001
    coarseValueImage = coarseDelayValue[i];

    // FineDelaySet
    fineDelayRegisterHigh =  (unsigned char) (fineDelayValue[i] / 256);           // C003
    fineDelayRegisterLow =   (unsigned char) (fineDelayValue[i] & 0x00FF);        // C002
    fineDelayStrobe = 1;                                                          // C004
    fineValueImage = fineDelayValue[i];
}

void DelayInit(void) {

  xdata unsigned long coarseResidual;
  xdata unsigned long fineResidual;

  digitNumber = 6;                  // default for AOFSR
  strcpy(delayString, "0.00000");  //

  timebaseReference = MIN_TB_REF*UNITSCALE;
  timebaseScale = DEF_TB_SCALE*UNITSCALE;
  timebaseRange = MAX_TB_RANGE*UNITSCALE;
  timebaseNumberOfSteps = MIN(ceil(timebaseRange/timebaseScale), MAXNUMSTEPS);
  timebaseCurrentStep = 0;

  delayValuesNeedToBeRecalculated = TRUE;    // calculate them
  sweepType = STOP;                          // haven't done anything yet

  coarseDelayValue[0] = CalculatedCoarseDelayNumber(0, &coarseResidual);
  fineDelayValue[0] = CalculatedFineDelayNumber(coarseResidual, &fineResidual);
  residual[0] = fineResidual;

  SetDelay(0);

  stepTriggerSlope = NEGATIVE;
  stepTriggerDelay = MIN_DELAY;
  stepTriggerSource = INTERNAL;

  sweepTriggerSlope = NEGATIVE;
  sweepTriggerDelay = MIN_DELAY;
  sweepTriggerSource = INTERNAL;

}
```

```
void EnableSweepTriggerInterrupt(void) {

  switch (sweepTriggerSlope) {
    case NEGATIVE:      // uses EXT3
      EXIF &= ~IE3_;  // Clear interrupt flag  //EXIF.5 = 0;
      EX3= 1;           // Enable interrupt      //EIE.1 = 1;
      break;
    case POSITIVE:      // uses EXT4
      EXIF &= ~IE4_;  // Clear interrupt flag  //EXIF.6 = 0;
      EX4 = 1;          // Enable interrupt      //EIE.2 = 1;
      break;
  }
}

void EnableStepTriggerInterrupt(void) {

  switch (stepTriggerSlope) {
    case NEGATIVE:  // uses EXT1
      IE1 = 0;        // Clear interrupt flag  //TCON.3 = 0;
      EX1 = 1;        // Enable interrupt      //IE.2 = 1;
      break;
    case POSITIVE:  // uses EXT2
      EXIF &= ~IE2_;// Clear interrupt flag  //EXIF.4 = 0;
      EX2 = 1;        // Enable interrupt      //EIE.0 = 1;
      break;
  }
}

void StartTimer(unsigned long desiredTimeout) {

    xdata union unionUnsignedLong stepDelay;


  ET0 = 0;              // Disable Timer 0 just in case was on //IE.1 = 0;
  TR0 = 0;              // Turn off timer while loading things  //TCON.4 = 0;
  TF0 = 0;              // Clear Timer 0 interrupt bit      //TCON.5 = 0;

  // timerDelay is the desiredTimeout in clocktics (or clock / divider)
  // either can be clock freq / 4 or clock freq / 12
  // clock frequency is in Mhz; since the desired timeout is in uS
  // the units cancels out and we get . . .

  stepDelay.all = (unsigned long) ( (float) desiredTimeout * CLOCKFREQ/DIVIDER);
  timerCount.byte.b1 = stepDelay.byte.b3;
  timerCount.byte.b0 = stepDelay.byte.b2;

  TH0 = ~stepDelay.byte.b1;   // Timer lowbyte
  TL0 = ~stepDelay.byte.b0;   // Timer highbyte
  TH0image = 0;               // reload with max delay for 32 bit counting
  TL0image = 0;               // 100h - 1 = FF and since count up use 00.

  ET0 = 1;                    // enable Timer0 interrupt      //IE.1 = 1;
  TR0 = 1;                    // Turn on timer 0              //TCON.4 = 1;
}

void ToggleAtoDStrobe(void) {  // use EXT0 interrupt pin P3.2.

//The trigger signal needs to be AT LEAST 10ns wide (either polarity)
//This routine assumes Rising-Edge Polarity
                 // Do we need to disable interrupts so that this signal doesn't
  P3_2 = 0;      // get caught high?
  P3_2 = 1;      // Like what is a minimum time for writing a high and then low?
  P3_2 = 0;      // hopefully greater than 10ns!!
                 // The A/D uses this signal to know when to begin an acquisition
}
```

```
void startSweepTrig(void) {

  sweeping = TRUE;
  timebaseCurrentStep = 0;

  switch(stepTriggerSource) {
    case INTERNAL:
      StartTimer(stepTriggerDelay);
      break;
    case EXTERNAL:
      EnableStepTriggerInterrupt();
      break;
    default:
      StartTimer(stepTriggerDelay);
      break;
  }

}

void startStepTrig(void) {

  if ( timebaseCurrentStep <= timebaseNumberOfSteps ) {
    // still within current sweep - send out new delay
    SetDelay(timebaseCurrentStep++);
    ToggleAtoDStrobe();
    switch (stepTriggerSource) {
      case INTERNAL:
        StartTimer(stepTriggerDelay);    // INT 0
        break;
      case EXTERNAL:
        EnableStepTriggerInterrupt();    // INT 1 and INT 2
        break;
      default:
        StartTimer(stepTriggerDelay);
        break;
    }
  }
  else { // sweep is completed.
    if (sweepType == SINGLE) {
        sweeping = FALSE;
        sweepType = STOP;      // only wanted a single sweep
    }
    else {
      switch(sweepTriggerSource) {
        case INTERNAL:
          sweeping = TRUE;
          timebaseCurrentStep = 0;
          StartTimer(sweepTriggerDelay);
          break;
        case EXTERNAL:
          sweeping = FALSE;
          EnableSweepTriggerInterrupt();
          break;
        default:
          sweeping = TRUE;
          timebaseCurrentStep = 0;
          StartTimer(sweepTriggerDelay);
          break;

      }
    }
  }
}
```

```
void timer0InterruptHandler (void) interrupt 1 {      // use registerbank 2 for interrupt

// Timer has been extended to 32 bits by tacking on 16 bits of
// timerCount onto the front.  Total count is timerCount:TH0:TL0 - see StartTimer(delay).

  ET0 = 0;  // disable interrupt FIRST
  TR0 = 0;  // Turn off timer 0 (don't want to be interrupted randomly) //TCON.4 = 0;
  TF0 = 0;  // Clear Timer 0 interrupt bit.  //TCON.5 = 0;

  if (timerCount.all-- <= 0) { // extended timer has timed out (can timerCount go NEGATIVE ?)
    if (!sweeping) {
      startSweepTrig();
    }
    else {                    // was waiting for a timer interrupt to begin a sweep
      startStepTrig();        // Start up step triggering
    }
  }
  else {                      // reload base timer and go again.
    TH0 = TH0image;           // saved from STARTTIMER
    TL0 = TL0image;
    ET0 = 1;          // enable Timer 0 interrupt      //IE.1 = 1;
    TR0 = 1;          // Turn on Timer 0               //TCON.4 = 1;
  }
}

// The step trigger signal is routed to both EXT1 and EXT2 so the one enabled determines the edge.
void ExternalInt1Handler (void) interrupt 2 {     // use registerbank 2 for interrupt

// Step trigger negative edge detector (enabled to use a NEG edge to trigger the next delay step)

  EX1 = 0;    // Disable interrupt          //IE.2 = 0;
  IE1 = 0;    // Clear interrupt flag       //TCON.3 = 0;

  startStepTrig();
}

void ExternalInt2Handler (void) interrupt 8 {     // use registerbank 2 for interrupt

// Step trigger positive edge detector (enabled to use a POS edge to trigger the next delay step)

  EX2 = 0;           // Disable interrupt         //EIE.0 = 0;
  EXIF &= ~IE2_;     // Clear interrupt flag      //EXIF.4 = 0;

  startStepTrig();
}

// The sweep trigger signal is routed to both EXT3 and EXT4 so the one enabled determines the edge.
void ExternalInt3Handler (void) interrupt 9 {     // use registerbank 2 for interrupt

// Sweep trigger negative edge detector  (enabled to use a NEG edge to trigger the next delay sweep)

  EX3= 0;            // Disable interrupt       //EIE.1 = 0;
  EXIF &= ~IE3_;     // Clear interrupt flag    //EXIF.5 = 0;

  startSweepTrig();
}

void ExternalInt4Handler (void) interrupt 10 {     // use registerbank 2 for interrupt

// Sweep trigger positive edge detector   (enabled to use a POS edge to trigger the next delay sweep)

  EX4 = 0;       // Disable interrupt     //EIE.2 = 0;
  EXIF &= ~IE4_;  // Clear interrupt flag    //EXIF.6 = 0;

  startSweepTrig();
}
```

```
//
// Global Variable Definitions for Remote Code , GLOBDEF.H
//
// Memory mapped I/O definitions
// EEPROM nonvolitile memory usage
// Extends from 8800 to 8FFF

xdata unsigned char NV_GPIBBoardRevision _at_ 0x8800;
xdata unsigned char NV_MicroFirmwareRevision _at_ 0x8801;
xdata unsigned char NV_CrystalFrequency    _at_ 0x8802;
xdata unsigned char NV_SpaceHolder1[17]  _at_ 0x8803;

//          Saved for expansion

xdata unsigned char NV_RS232Baudrate       _at_ 0x8814;
    // 0:300,1:1200,2:2400,3:4800,4:9600,5:19200
xdata unsigned char NV_GPIBAddress         _at_ 0x8815;
xdata unsigned char NV_ActiveDisplayMode  _at_ 0x8816;
xdata unsigned char NV_SpaceHolder2[745]  _at_ 0x8817;
// size without fine array
//xdata unsigned char NV_SpaceHolder2[233]    _at_ 0x8817;
//xdata unsigned long NV_fineDelayArray[128]   _at_ 0x8900;
xdata unsigned long NV_coarseDelayArray[256]  _at_ 0x8B00;
xdata float  NV_fineCalCoeff[64]      _at_ 0x8F00;

//            Saved for expansion
//                     0x8FFF;
// Display Board

xdata unsigned char DisplayVersion              _at_ 0xE800
    /* Display version number (read only) */
xdata unsigned char Buttons                     _at_ 0xE880;
    /* Display switches (read only)
            Bit    .0 Local (Latched)
                   .1 GPIB (Momentary)
                   .2 RS232 (Momentary)
                   .3 Display (Latched)
    */
xdata unsigned char LocalButton_RESET       _at_ 0xE900;
    /* Accessing this location resets local Switch FF */
xdata unsigned char DisplayButton_RESET    _at_ 0xE980;
    /* Accessing this location resets display switch FF */
xdata signed   char EncoderRegister       _at_ 0xEA00;
    /* encoder read location */

xdata unsigned char DisplayControl        _at_ 0xEA80;
    /* Display_Control byte
            Bit    .0 : !RST of Encoder
                   .1 : !LeftDisplay_RESET
                   .2 : !RightDisplay_RESET
                   .3 : AddressedLED
                   .4 : RemoteLED
    */
xdata unsigned char DisplayControlMask;
/* Copy of DisplayControl register */
xdata unsigned char LeftDisplayFlash_0  _at_ 0xEB00;
xdata unsigned char LeftDisplayFlash_1  _at_ 0xEB01;
xdata unsigned char LeftDisplayFlash_2  _at_ 0xEB02;
xdata unsigned char LeftDisplayFlash_3  _at_ 0xEB03;
xdata unsigned char LeftDisplayFlash_4  _at_ 0xEB04;
xdata unsigned char LeftDisplayFlash_5  _at_ 0xEB05;
xdata unsigned char LeftDisplayFlash_6  _at_ 0xEB06;
xdata unsigned char LeftDisplayFlash_7  _at_ 0xEB07;
xdata unsigned char LeftDisplayCnt       _at_ 0xEB30;
xdata unsigned char LeftDisplay_0   _at_ 0xEB38;
xdata unsigned char LeftDisplay_1   _at_ 0xEB39;
xdata unsigned char LeftDisplay_2   _at_ 0xEB3A;
```

```
xdata unsigned char LeftDisplay_3   _at_ 0xEB3B;
xdata unsigned char LeftDisplay_4   _at_ 0xEB3C;
xdata unsigned char LeftDisplay_5   _at_ 0xEB3D;
xdata unsigned char LeftDisplay_6   _at_ 0xEB3E;
xdata unsigned char LeftDisplay_7   _at_ 0xEB3F;

xdata unsigned char RightDisplayFlash_0     _at_ 0xEB80;
xdata unsigned char RightDisplayFlash_1     _at_ 0xEB81;
xdata unsigned char RightDisplayFlash_2     _at_ 0xEB82;
xdata unsigned char RightDisplayFlash_3     _at_ 0xEB83;
xdata unsigned char RightDisplayFlash_4     _at_ 0xEB84;
xdata unsigned char RightDisplayFlash_5     _at_ 0xEB85;
xdata unsigned char RightDisplayFlash_6     _at_ 0xEB86;
xdata unsigned char RightDisplayFlash_7     _at_ 0xEB87;
xdata unsigned char RightDisplayCnt       _at_ 0xEBB0;
xdata unsigned char RightDisplay_0        _at_ 0xEBB8;
xdata unsigned char RightDisplay_1        _at_ 0xEBB9;
xdata unsigned char RightDisplay_2        _at_ 0xEBBA;
xdata unsigned char RightDisplay_3        _at_ 0xEBBB;
xdata unsigned char RightDisplay_4        _at_ 0xEBBC;
xdata unsigned char RightDisplay_5        _at_ 0xEBBD;
xdata unsigned char RightDisplay_6        _at_ 0xEBBE;
xdata unsigned char RightDisplay_7        _at_ 0xEBBF;

// Slot ID's
// C000 - E7FF

//xdata unsigned char Slot1ID        _at_ 0xC000;
//xdata unsigned char Slot2ID        _at_ 0xC800;
//xdata unsigned char Slot3ID        _at_ 0xD000;
//xdata unsigned char Slot4ID        _at_ 0xD800;
//xdata unsigned char Slot5ID        _at_ 0xE000;

// Switch status and debouncing flags
bit       LocalButton_Status;
bit       DisplayButton_Status;
bit       GPIBButton_Status;
bit       RS232Button_Status;
bit       _LOCAL_BUTTON_PUSHED;
bit       _DISPLAY_BUTTON_PUSHED;

// Enumerated types

enum type_PicomotorState {
    LOCAL,
    REMOTE,
    CHANGINGMASTERMENU,
    CHANGINGCOMMENU,
    CHANGINGCOMVALUE,
    ERROR,
    CHANGINGDELAYVALUE,     // afosr mod
    SWEEPING             // afosr
    };
data enum type_PicomotorState CurrentState, NextState,
PreviousState;

enum type_EncoderFunction {
    SWITCHNUMBER,
    DISPLAYMODE,
    GPIBADDRESS,
    BAUDRATE,
    NOFUNCTION,
    COMSELECT,
    MENUNUMBER,
    DELAYDIGIT      // afosr mod
    };
data enum type_EncoderFunction EncoderFunction;
```

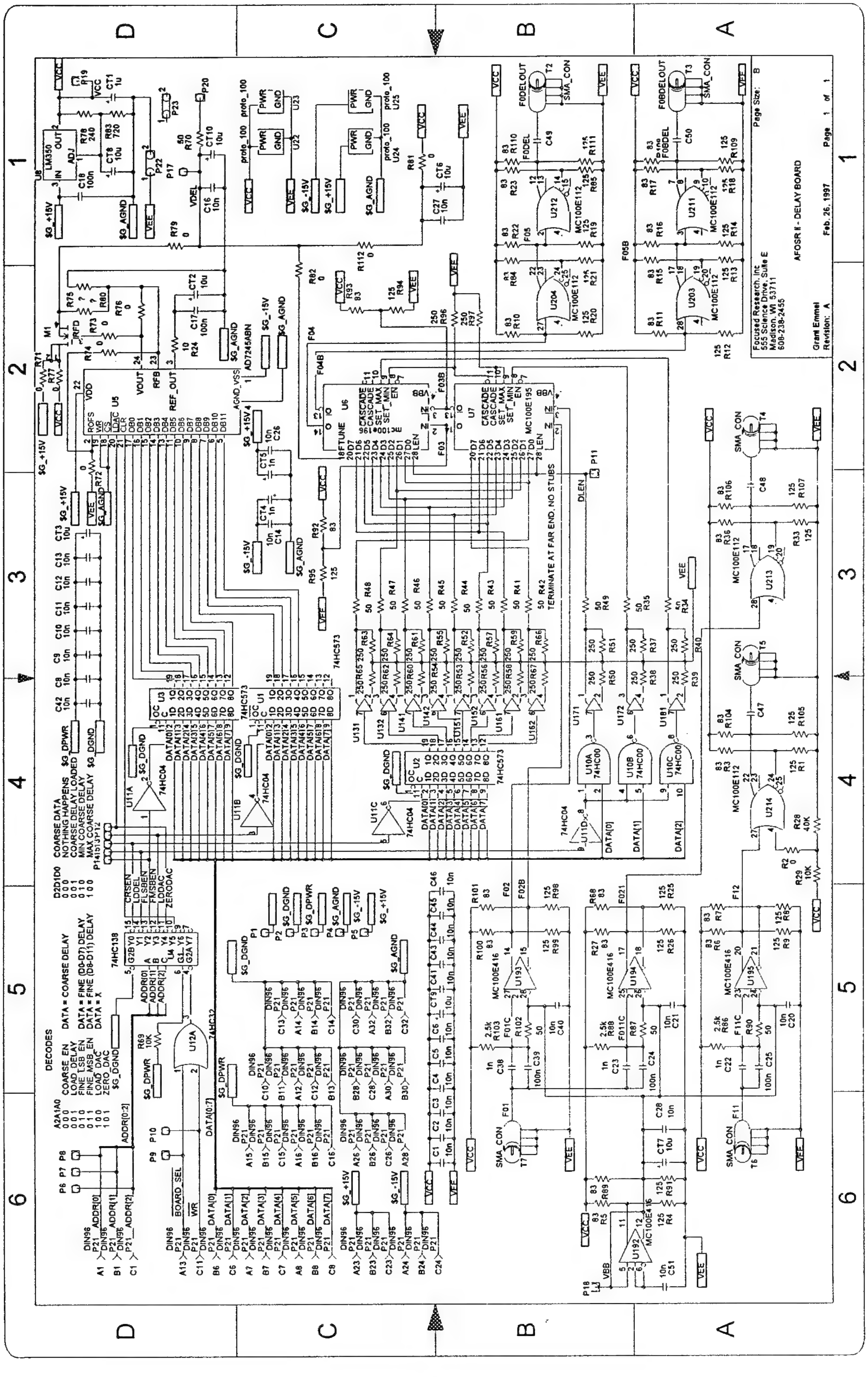
```
enum type_DisplayType {     // used to select active port */
    _ERRORDISP,
    _SWITCHDISP,
    _LOCALDISP,
    _GPIBDISP,
    _RS232DISP,
    _KEYBOARDDISP,
    _COMCHOICEDISPLAY,
    _MENU0,
    _DELAYEDIT,         // aofsr mod
    _DELAY,         // aofsr mod
    _SWEEPDISPLAY
                    };
data enum type_DisplayType DisplayType;

enum type_ComPort {         /* used to select active port */
    UNDEFINED,
    GPIB,
    SERIAL
    };
data enum type_ComPort ComPort;

// Encoder related variables

xdata signed char         Encoder;
    /* realtime encoder count updated in TIMER2 interrupt handler */
xdata unsigned long       Nothing;
    /* Dummy place holder for zero or ignored encoder updates */
xdata unsigned longSwitch_Number;
    /* Holds the linear number of the switch currently selected */
xdata unsigned longGPIB_Address_Image;
    /* Holds the GPIB address as it's being updated by encoder.  */
xdata unsigned longRS232_Baudrate_Image;
            /* holds baudrate as it's being updated by encoder */
xdata float        EncoderTotal;
    /* using float for overflow before max/min saturation enforced */
data unsigned char  EncoderNotZero;
    /* flag set in TIMER2 interrupt handler when encoder non zero */
data unsigned int    EncoderZeroCounter;
    /* counter that decrements every sample without encoder */
    /* reading gives a good look/feel to the track mode operation */
xdata unsigned long *   xdata EncoderTargetValuePtr;
    /* points to value to be updated by encoder */
xdata unsigned long *   xdata DisplayValuePtr;
    /* points to value to be displayed on Wavelength Display */

// general stuff

data unsigned char ch;
xdata unsigned int  unitsValue;
xdata float         numberValue;
xdata unsigned long      longNumberValue;
xdata char          outbuf[256];
xdata char          dispbuf[50];
xdata unsigned charError;
xdata char          ErrorTxt[17];
data unsigned char  CurrentInput, PreviousInput;

// computer control globals
xdata unsigned charCOMError;
bit                 NEW_GPIB_ADDRESS;
xdata char global_token[20];
    /* holds token so can use it as a channel name */

// serial handler globals
```

```
bit                 serialCommandReceived;
bit                 OutputBufferEmpty;
xdata unsigned charoutputPointer;
xdata unsigned charinputPointer;
bit                 ADDRESSED_LED;

// realtime.c
//
xdata unsigned int  AddressedLEDBlink;
xdata unsigned long      DisplayModeValue;
xdata unsigned charflashValue;
    // index to array of flash masks to control blinking of
display
xdata unsigned long      ConnectorDisplayChoice;
xdata unsigned long      ComChoice;
bit                 _comflag;
    // set and cleared by Com button
bit                 _displayflag;
    // set and cleared by display mode button
code unsigned char leftFlashArray[] = {
    0x00,   // 0 _FlashNothing
    0x01,   // 1 _FlashDisplayA
    0x00,   // 2 _FlashDisplayB
    0x00,   // 3 _FlashDisplayC
    0x00,   // 4 _FlashDisplayConnector
    0xF0,   // 5 _FlashReturn
    0x07,   // 6 _FlashChooseChanAType
    0x00,   // 7 _FlashChooseChanBType
    0x00,   // 8 _FlashChooseChanCType
    0x03,   // 9 _FlashRS232
    0x00,   // 10 _FlashGPIB
    0x00,   // 11 _FlashRS232Value
    0x00,   // 12 _FlashGPIBValue
    0x00,   // 13 _FlashPicoTypeOption
    0x38,   // 14 _FlashComMenu
    0x07,   // 15 _FlashDispMenu
            // Not used
    0x00,   // 16 _FlashPicoMenu
    0xF0,   // 17 _FlashComExit
    0x7F,   // 18 _FlashDisplayDefault
            // Not used
    0x00,   // 19 _FlashDisplayUser
            // Not used
    0x00,   // 20 _FlashRS232ValueBrackets
    0x00};  // 21 _FlashGPIBValueBrackets

code unsigned char rightFlashArray[] = {
    0x00,   // 0 _FlashNothing
    0x00,   // 1 _FlashDisplayA
    0x40,   // 2 _FlashDisplayB
    0x10,   // 3 _FlashDisplayC
    0x07,   // 4 _FlashDisplayConnector
    0x00,   // 5 _FlashReturn
    0x00,   // 6 _FlashChooseChanAType
    0x70,   // 7 _FlashChooseChanBType
    0x07,   // 8 _FlashChooseChanCType
    0xE0,   // 9 _FlashRS232
    0x0F,   // 10 _FlashGPIB
    0xFF,   // 11 _FlashRS232Value
    0xFF,   // 12 _FlashGPIBValue
    0xFF,   // 13 _FlashPicoTypeOption
    0x00,   // 14 _FlashComMenu
    0x80,   // 15 _FlashDispMenu
            // Not used
    0x3C,   // 16 _FlashPicoMenu
    0x00,   // 17 _FlashComExit
    0x00,   // 18 _FlashDisplayDefault
```

```
          // Not used anymore
    0x1E,   // 19 _FlashDisplayUser
          // Not used anymore
    0x41,   // 20 _FlashRS232ValueBrackets
    0x09};  // 21 _FlashGPIBValueBrackets

xdata unsigned long      MenuChoice;

xdata    float Total;
char xdata  stringValue[80];
xdata    char     token[80];
xdata    unsigned char SlotID;
xdata    char     OPCFlag;
bit      serialTransmissionInProgress;
xdata char  commandbuffer[256];
xdata char  * xdata commandstream;

// New or modified constants/variables for AOFSR
xdata unsigned char digitNumber;
xdata float newdelay;
xdata float delay;
xdata float coarse;
xdata float fine;
xdata unsigned char fineValueHighImage;
xdata unsigned char fineValueLowImage;
xdata unsigned char coarseValueImage;
xdata unsigned int fineValueImage;
xdata unsigned char coarseDelayRegister     _at_ 0xC000;
xdata unsigned char coarseDelayStrobe  _at_ 0xC001;
xdata unsigned char fineDelayRegisterHigh _at_ 0xC003;
xdata unsigned char fineDelayRegisterLow    _at_ 0xC002;
xdata unsigned char fineDelayStrobe       _at_ 0xC004;
xdata char displayDigitChar;
xdata char delayString[9];
xdata unsigned long delayDigitValue;
xdata unsigned int DimTimeCount;
bit NowDimFlag;

// type def for union accessing of bytes inside longs and integers

typedef union unionUnsignedLong    {
          unsigned long all;
    struct   { unsigned char b3,b2,b1,b0;    } byte;
    };
typedef union unionUnsignedInt {
    unsigned int all;
    struct   { unsigned char b1,b0;  } byte;
    };
typedef union unionFloat         {
    float all;
    struct   { unsigned char b3,b2,b1,b0;    } byte;
    };

enum type_SweepType {
    STOP,
    SINGLE,
    RUN
    };
xdata enum type_SweepType sweepType;

enum type_TriggerSource {
    INTERNAL,
    EXTERNAL
    };
xdata enum type_TriggerSource stepTriggerSource;
xdata enum type_TriggerSource sweepTriggerSource;
```

```
enum type_CalData {
    COARSE,
    FINE };

enum type_Slope {
    NEGATIVE,
    POSITIVE };

xdata enum type_Slope stepTriggerSlope;
xdata enum type_Slope sweepTriggerSlope;

bit  sweeping;
bit delayValuesNeedToBeRecalculated;

xdata unsigned long currentDelay;
xdata unsigned int  timebaseCurrentStep;

xdata unsigned long timebaseReference;
xdata unsigned long timebaseScale;
xdata unsigned long timebaseRange;
xdata unsigned int   timebaseNumberOfSteps;

xdata unsigned int coarseDelayValue[MAXNUMSTEPS];
xdata unsigned int fineDelayValue[MAXNUMSTEPS];
xdata unsigned int residual[MAXNUMSTEPS];

xdata unsigned long stepTriggerDelay;
xdata unsigned long sweepTriggerDelay;

xdata unsigned char TH0image;
xdata unsigned char TL0image;

xdata union unionUnsignedInt timerCount;
```

Appendix 3 – Delay Board Schematic



Focused Research, Inc
555 Science Drive, Suite E
Madison, WI 53711
608-238-2455

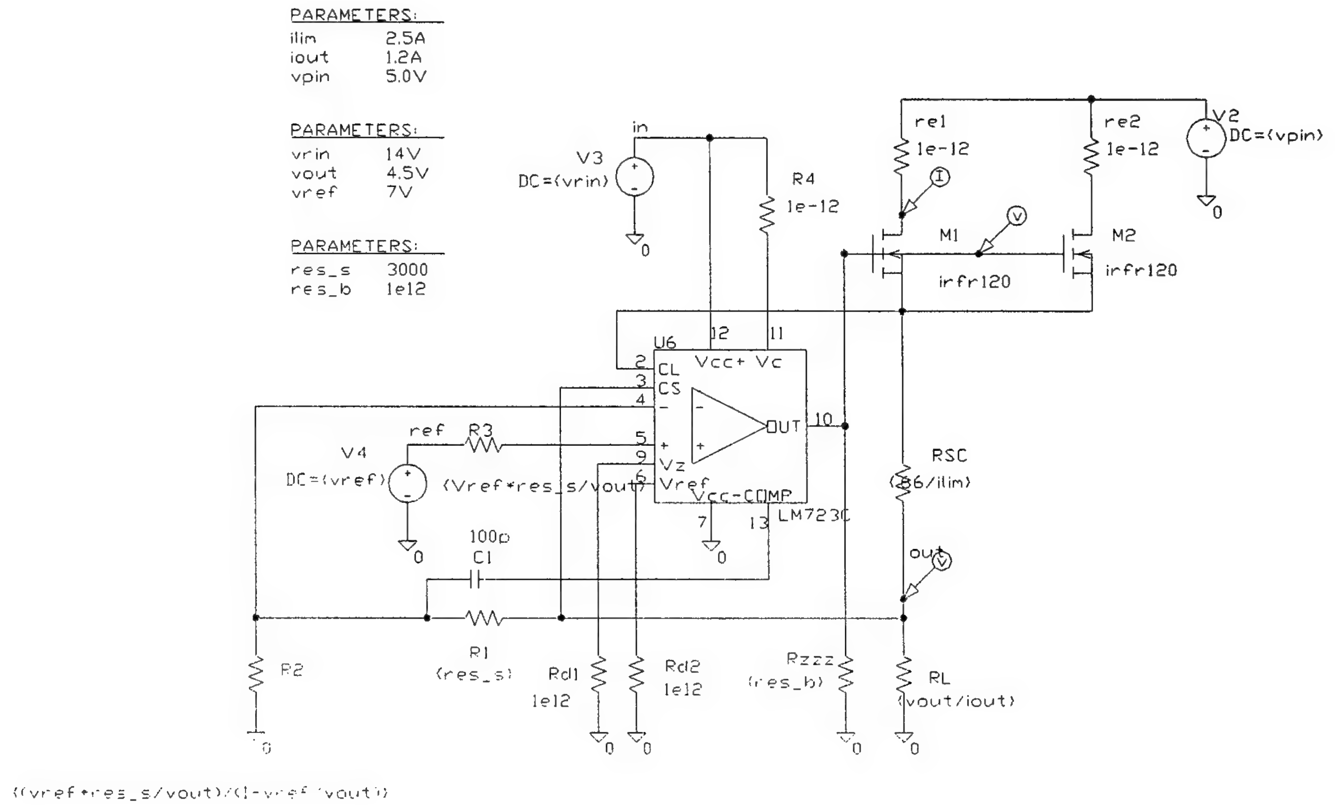
Grant Emmel
Revision: A

AFOSR II - DELAY BOARD

Feb. 26, 1997

Page Size: B

Page 1 of 1



PARAMETERS:	
ilim	2.5A
iout	1.2A
vpin	5.0V

PARAMETERS:	
vrin	14V
vout	4.5V
vref	7V

PARAMETERS:	
res_s	3000
res_b	1e12

Focused Research
555 Science Drive
Madison, WI 53711
608-238-2455

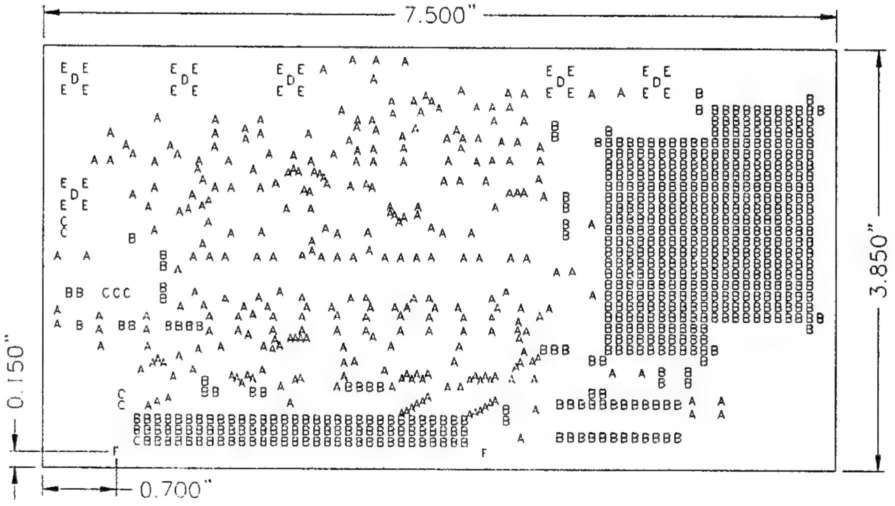
Page Size: A

Revision fill_in_rev fill_in_datePage 1 of 1

Appendix 4 – Delay Board PCB layout

FABRICATION SPECIFICATIONS AND NOTES	
MATERIAL	☒ FR4 EPOXY GLASS
FINISHED MATERIAL THICKNESS MEASURED OVER THE GOLD CONTACTS WHEN APPLICABLE	□ .062" +.005",-.004" ☒ .063" +.008",-.008"
COPPER THICKNESS PLATED THROUGH HOLES	☒ .001" MINIMUM
COPPER THICKNESS INTERNAL LAYERS	□ 1/2 OZ. ☒ 1 OZ. □ 2 OZ. GND AND VCC MUST BE 1OZ.
COPPER THICKNESS EXTERNAL LAYERS	□ 1/2 OZ. ☒ 1 OZ. □ 2 OZ.
TIN LEAD REFLOW MINIMUM THICKNESS	.0001" IN PLATED THRU HOLES. .0004" ON PADS. .00005" AT KNEE JOINT. SMT PADS .0002" MIN. .0004" MAX.
TYPE OF REFLOW	☒ I/R ☒ HOT AIR □ HOT OIL
MINIMUM ANNULAR RING (SOLDER SIDE) NO BREAKOUT ALLOWED	☒ .001" □ .005"
CONDUCTOR WIDTH	☒ .010" ☒ .020" □ .025" (INNER LAYERS) □ .016" ☒ .055" TOLERANCE: +.002" -.002"
BOW AND TWIST	☒ PER IPC □ .007" PER INCH
SOLDER MASK MASK MUST PASS IPC-SM-840A	☒ WET FILM- PC401, PC501, SP1010 □ PHOTO IMAGEABLE- LEA RONAL, TAIYO, CIBA GEIGY ☒ SMOBC- WITH PHOTO IMAGEABLE ONLY
SILKSCREENING (EPOXY BASED INK ONLY)	☒ COMPONENT SIDE □ SOLDER SIDE □ NONE
SILKSCREEN COLOR	☒ WHITE □ YELLOW □ RED
ALL BOARDS TO BE ELECTRICALLY TESTED	□ YES ☒ NO TEST STAMP ON SOLDER SIDE ONLY.
MANUFACTURERS LOGO ON SOLDER SIDE	☒ YES □ NO ☒ 94V-0 □ 94V-2
BOARDS TO BE INDIVIDUALLY WRAPPED	□ YES ☒ NO

CONDUCTIVE LAND PATTERN SHALL NOT BE MISREGISTERED FROM FRONT TO BACK BY MORE THAN .005".
NO SOLDER MASK ALLOWED ON PADS OR IN PLATED THROUGH HOLES.
NO EXPOSED TRACES AFTER MASKING.



LAYER ASSIGNMENTS

LAYER 1 (COMPONENT SIDE)
LAYER 2 (Ground Plane)
LAYER 3 (Power Plane)
LAYER 4 (Solder Side)

HOLE SIZES

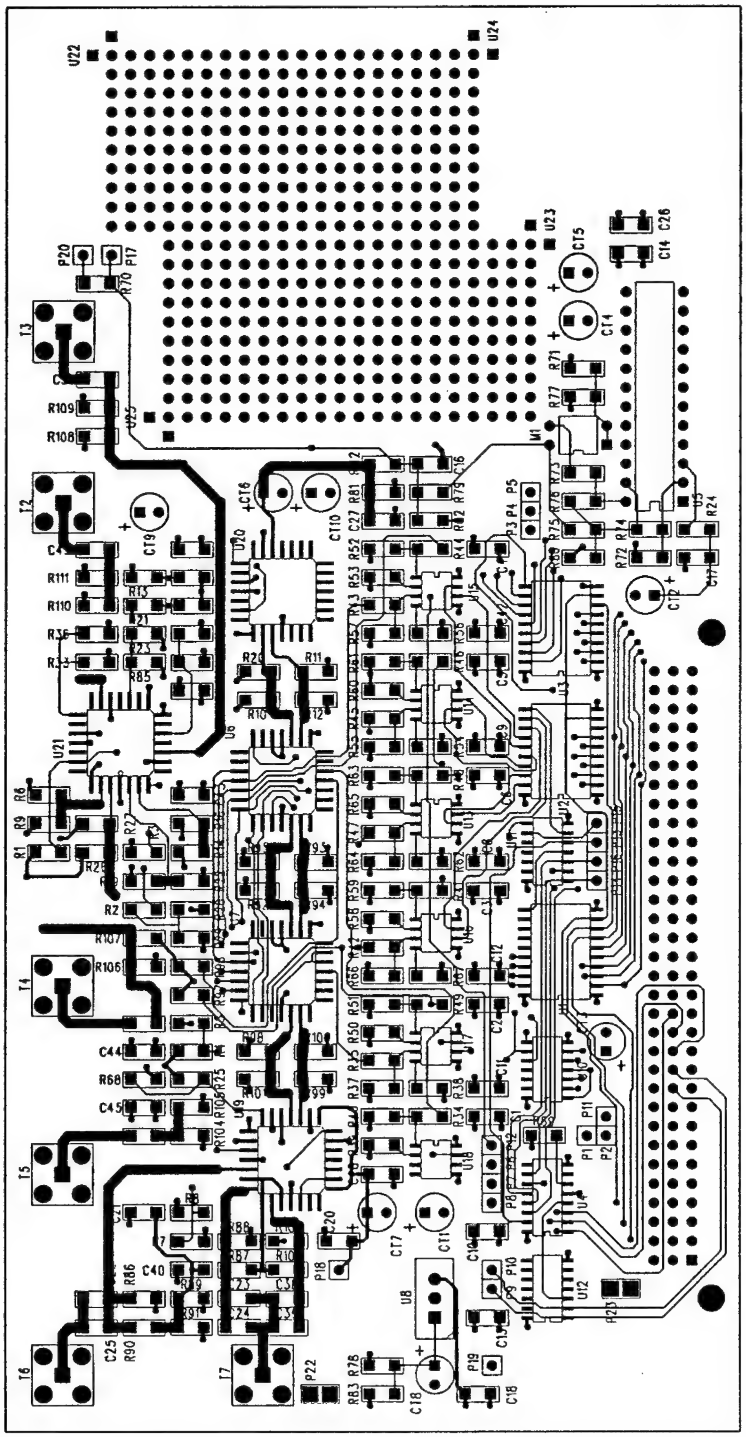
A = .018" +/- .003" (297 PLACES)
B = .035" +/- .003" (571 PLACES)
C = .040" +/- .003" (8 PLACES)
D = .066" +/- .004" (6 PLACES)
E = .080" +/- .005" (24 PLACES)
F = .100" +/- .005" (2 PLACES)

ARTWORK REVISED				
REVISIONS	REV	REV BY	DATE	ECO/APPVL

New Focus

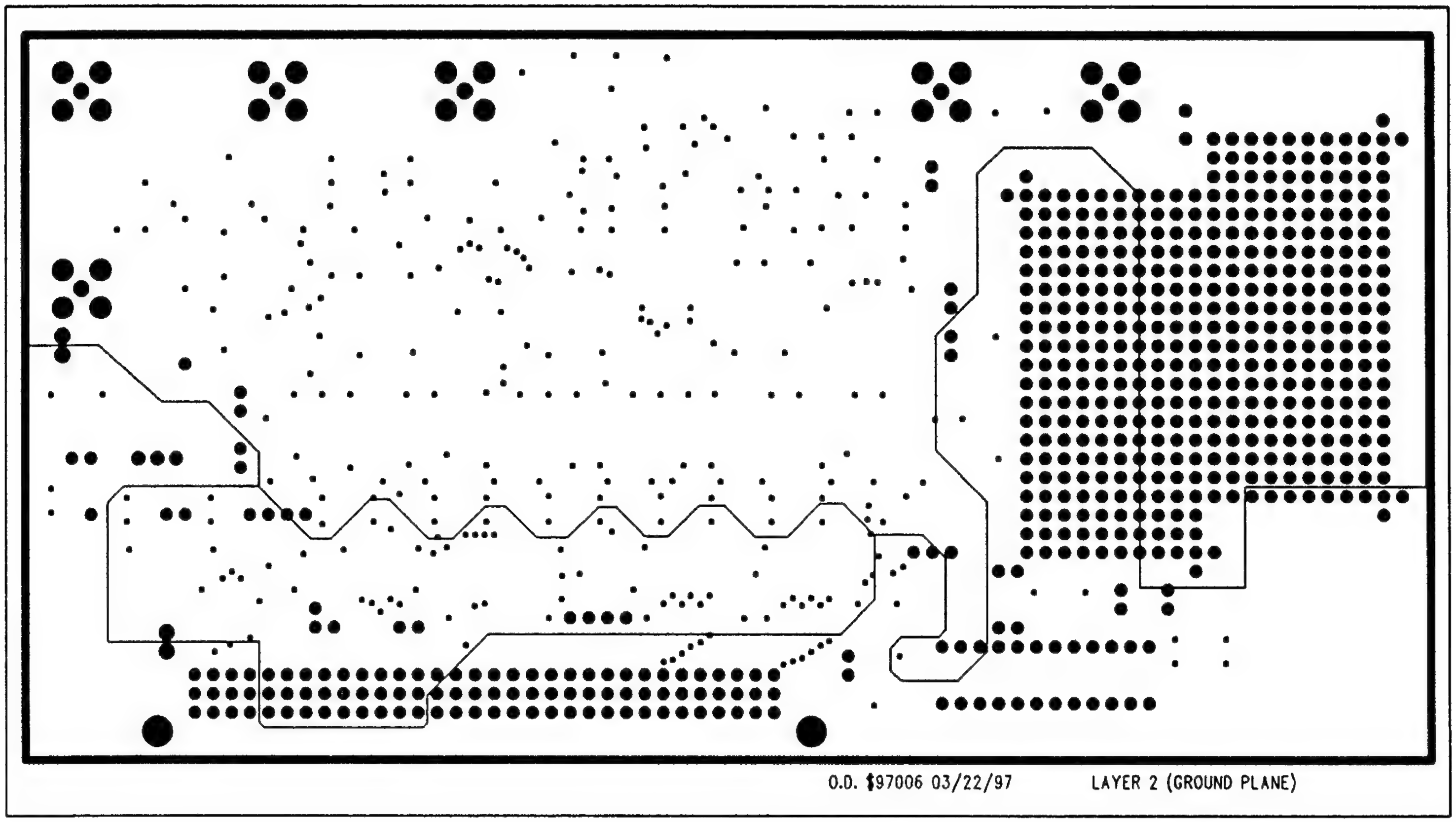
Madison, Wisconsin

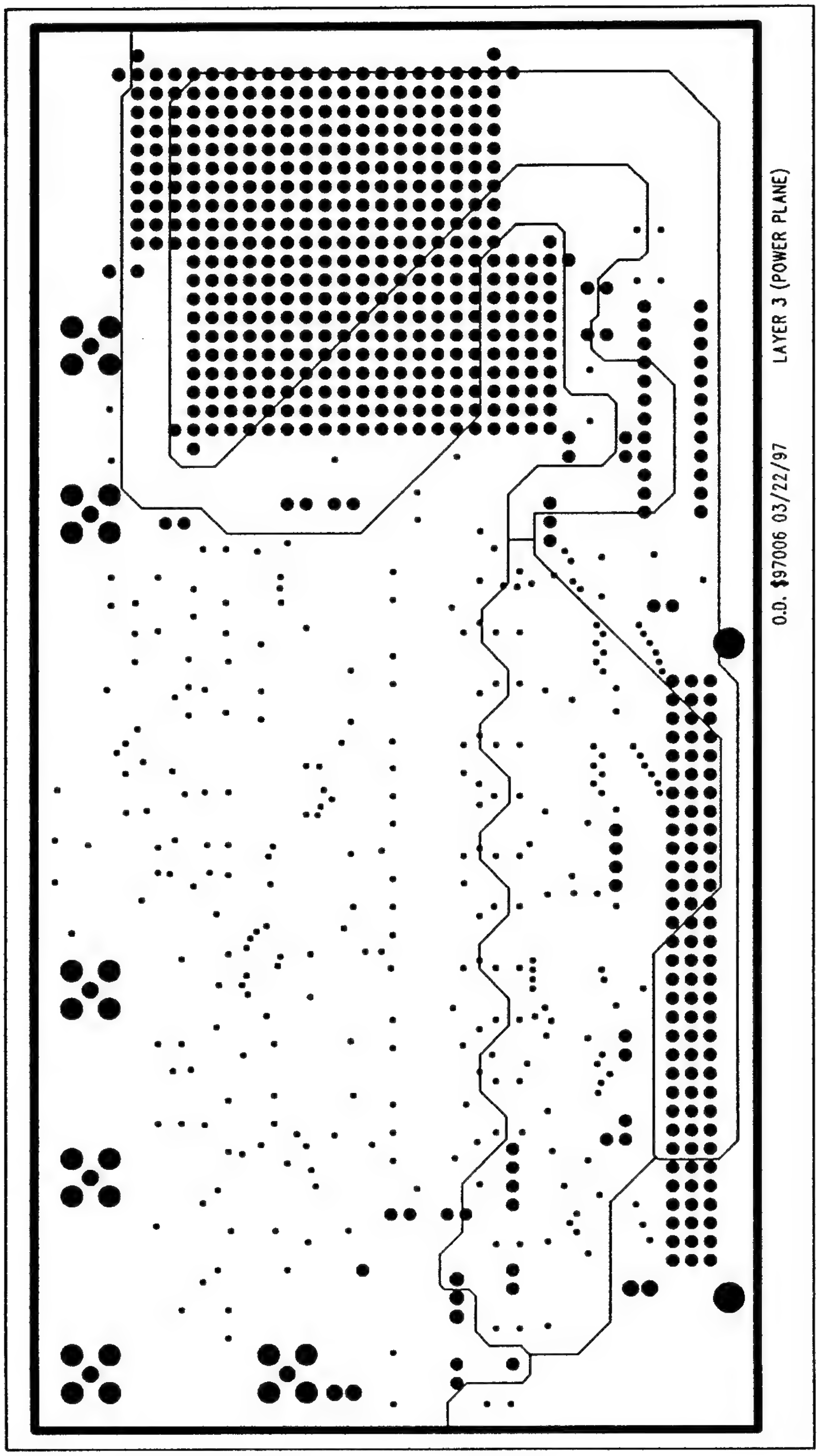
DWN: RHM	CKD: kpm	TOL: ANGLE ±1° 2 PL. DEC. ± .03 3 PL. DEC. ± .015
DATE: 3/22/97	SCALE: FULL	
TITLE: Delay Board		
SHEET: 1 OF 1	COPYRIGHT: 1995	DWG. NO: $97006

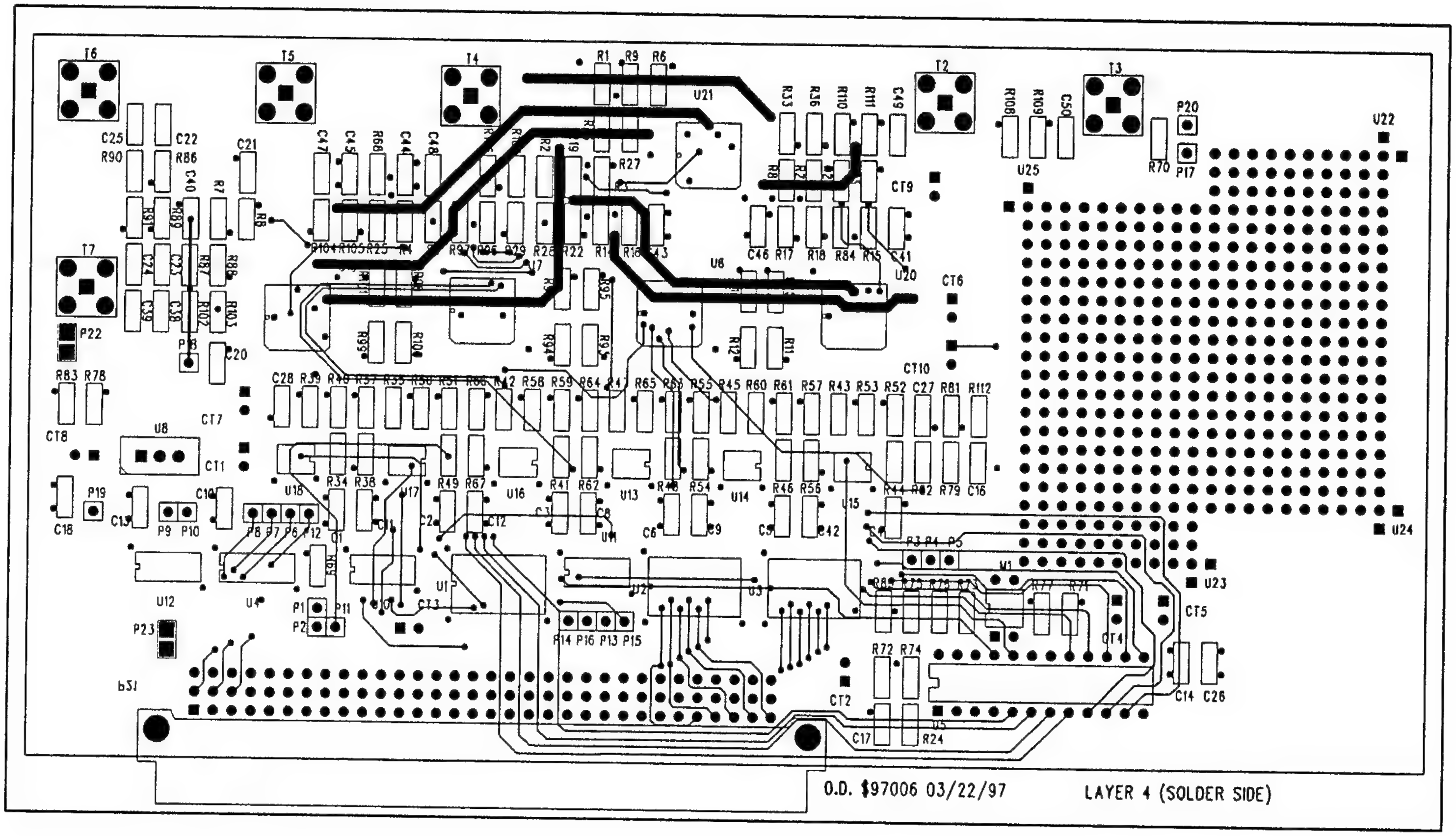


LAYER 1 (COMPONENT SIDE)

O.D. $97006 03/22/97


O.D. $97006 03/22/97
LAYER 2 (GROUND PLANE)


O.D. $97006 03/22/97
LAYER 3 (POWER PLANE)

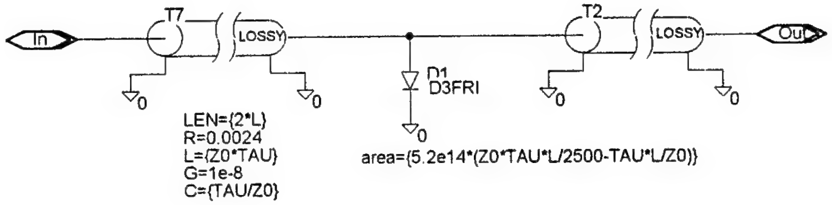


O.D. $97006 03/22/97

LAYER 4 (SOLDER SIDE)

Appendix 5 – NLTL Schematics

UnitNLTL2.sch
UnitNLTL3.sch
UnitNLTL4.sch
Predriver2.sch
Predriver1.sch

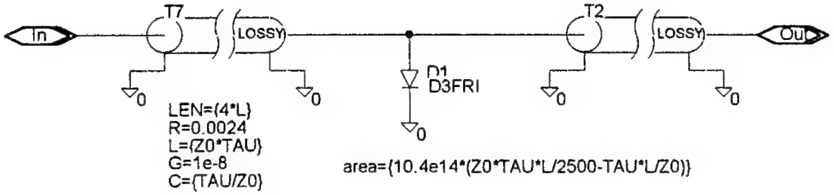


UNIT NLTL 2

Focused Research
555 Science Drive
Madison, WI 53711
608-238-2455

Page Size: A

Revision: fill_in_rev fill_in_date Page 1 of 1

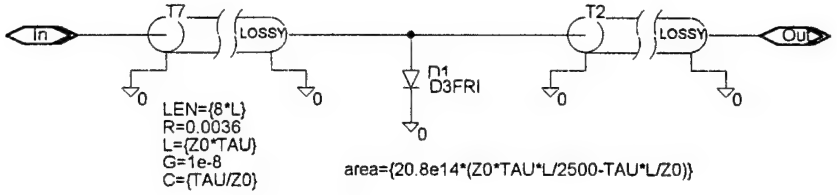


UNIT NLTL 3

Focused Research
555 Science Drive
Madison, WI 53711
608-238-2455

Page Size: A

Revision: fill_in_rev fill_in_date Page 1 of 1

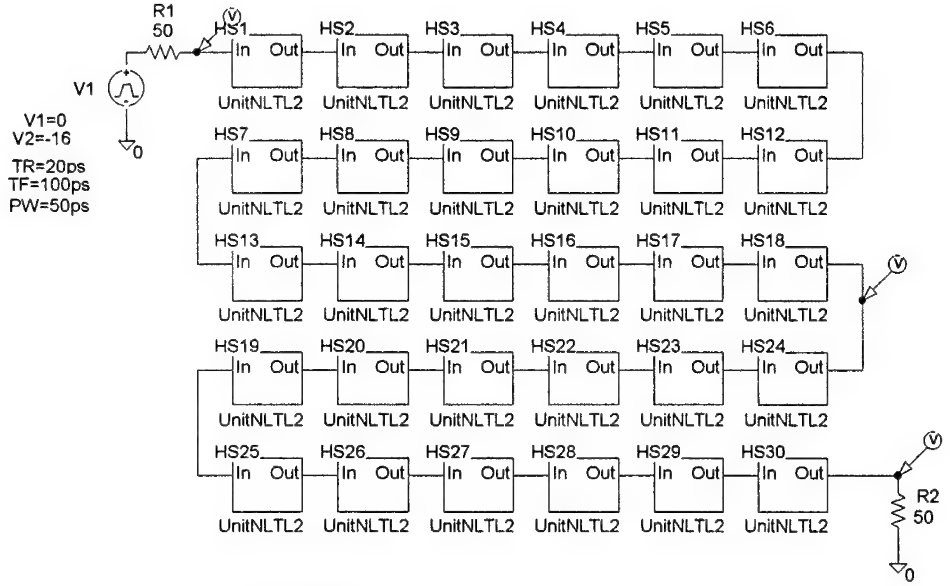


UNIT NLTL 4

Focused Research
555 Science Drive
Madison, WI 53711
608-238-2455

Page Size: A

Revision: fill_in_rev fill_in_date Page 1 of 1



PARAMETERS:

Z0	90
TAU	0.01ps
L	60

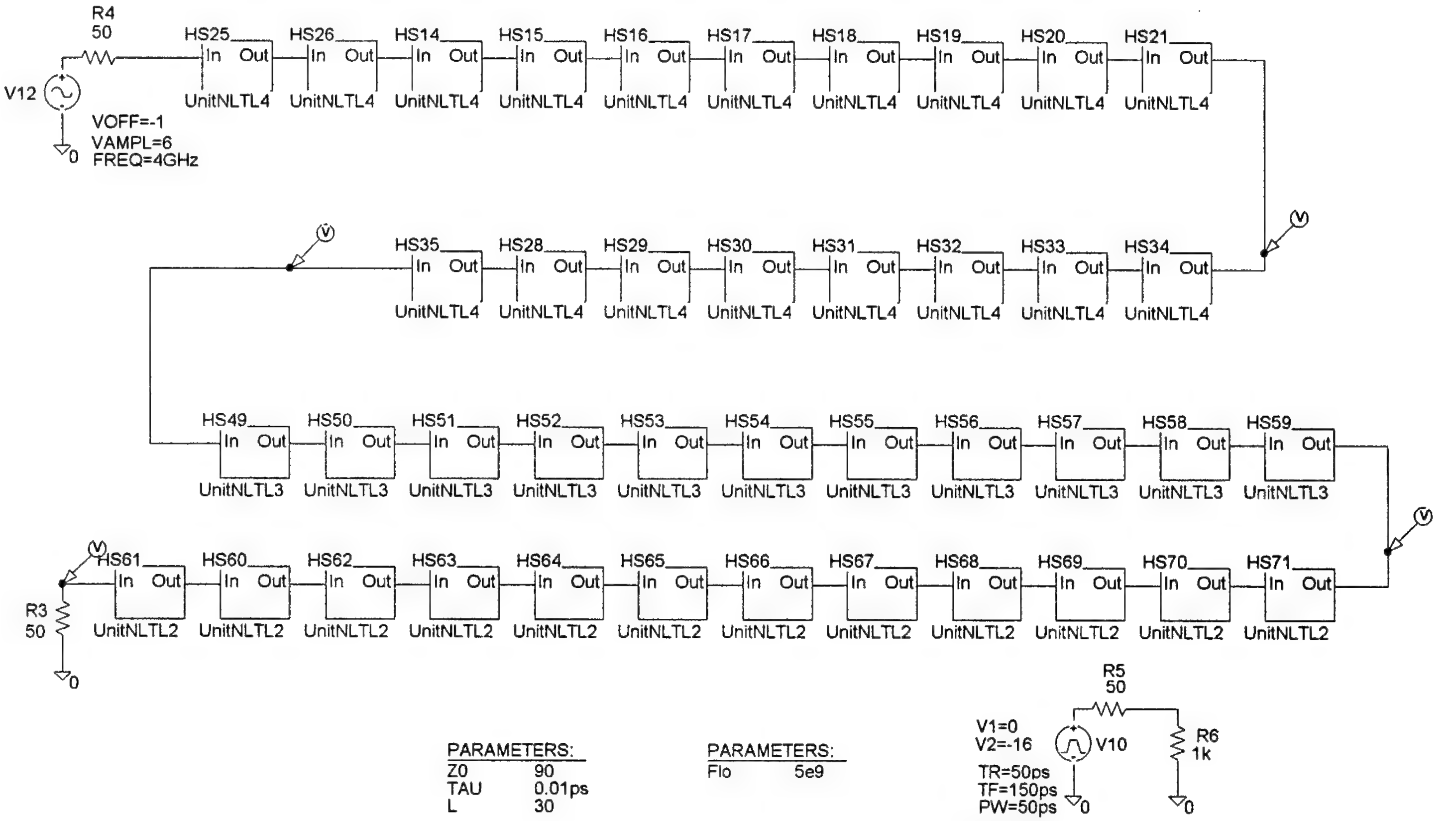
NLTL 2 . sch

(PRE-DRIVER 1 OR NLTL1 layout)

Focused Research
555 Science Drive
Madison, WI 53711
608-238-2455

Page Size: A

Revision: fill_in_rev fill_in_date Page 1 of 1



Focused Research
555 Science Drive
Madison. WI 53711
608-238-2455

Page Size: A

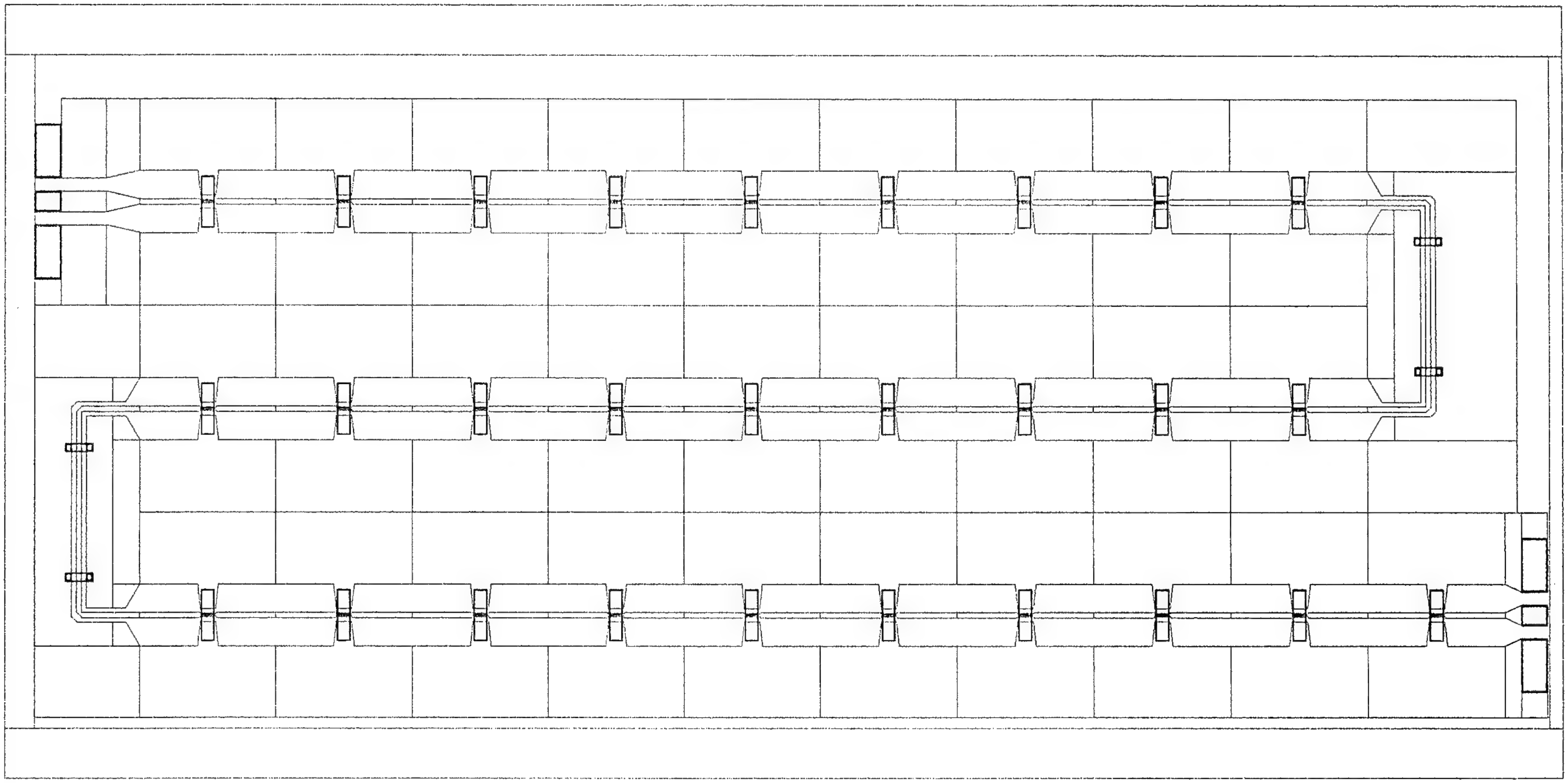
Revision: fill_in_rev fill_in_date Page 1 of 1

NLTLa.sch
(Pre-driver 2 or NLTL2 layout)

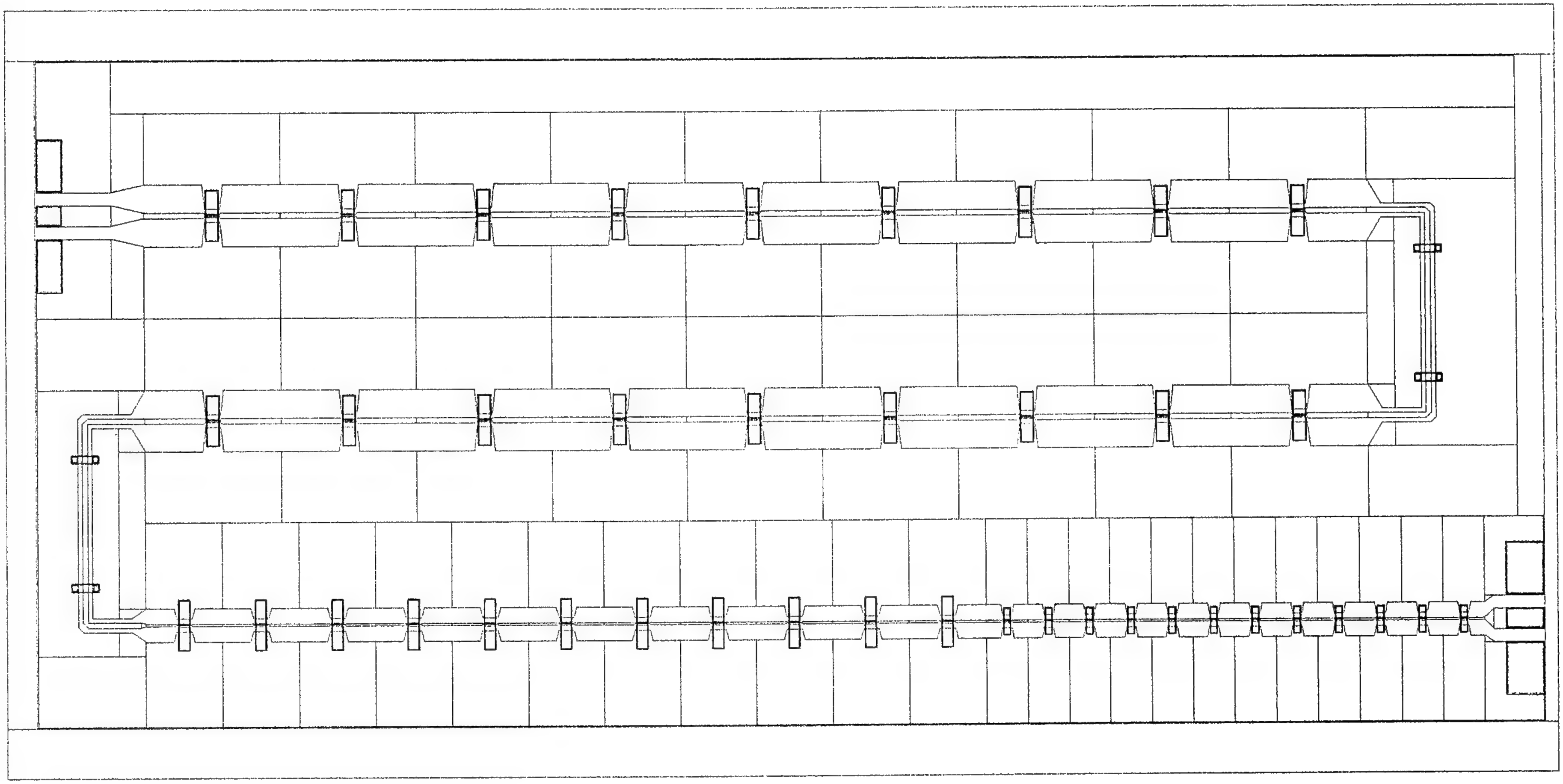
Appendix 6 – NLTL Physical Layout

Pre-driver 1 (3000x6000um)
Pre-driver 2 (3000x6000um)

PRE-DRIVER 1

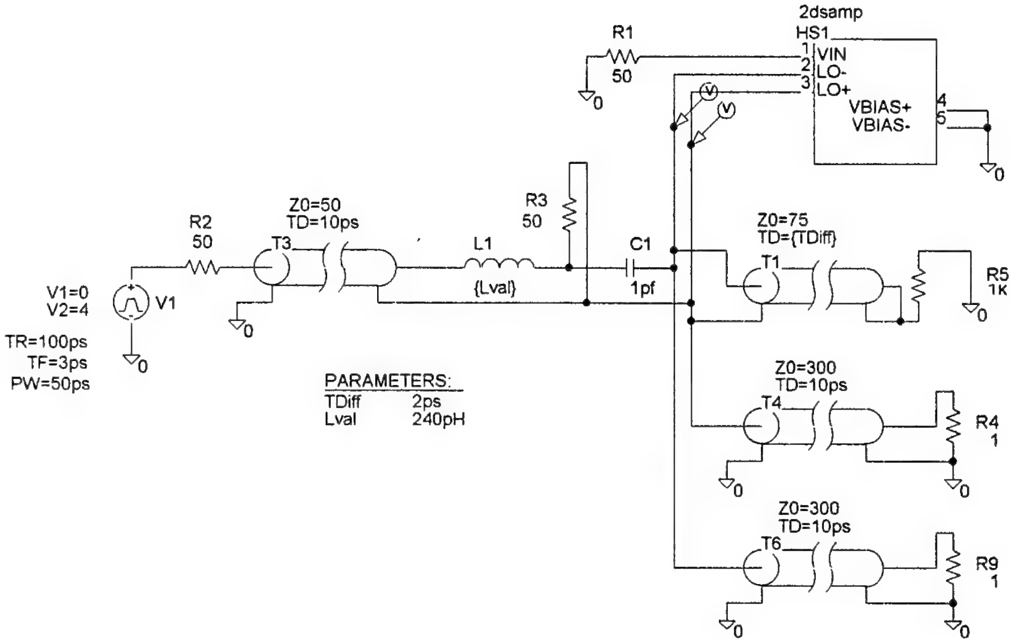


PRE-DRIVER 2



Appendix 7 – Sampler Schematic

SamplerTest.sch
2diodesampler.sch

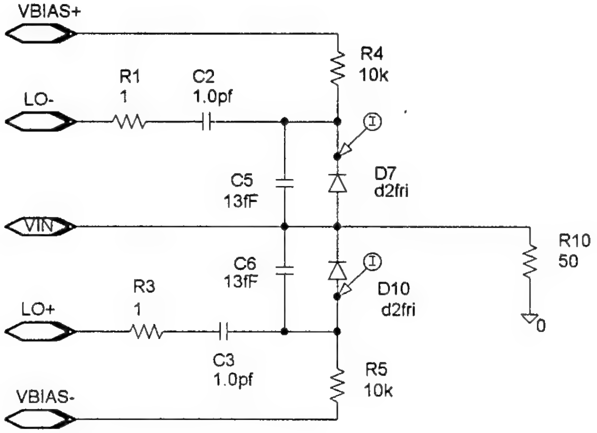


Sampler1test.sch

Focused Research
555 Science Drive
Madison, WI 53711
608-238-2455

Page Size: A

Revision: fill_in_rev fill_in_date Page 1 of 1



2 diode sampler

Focused Research
555 Science Drive
Madison, WI 53711
608-238-2455

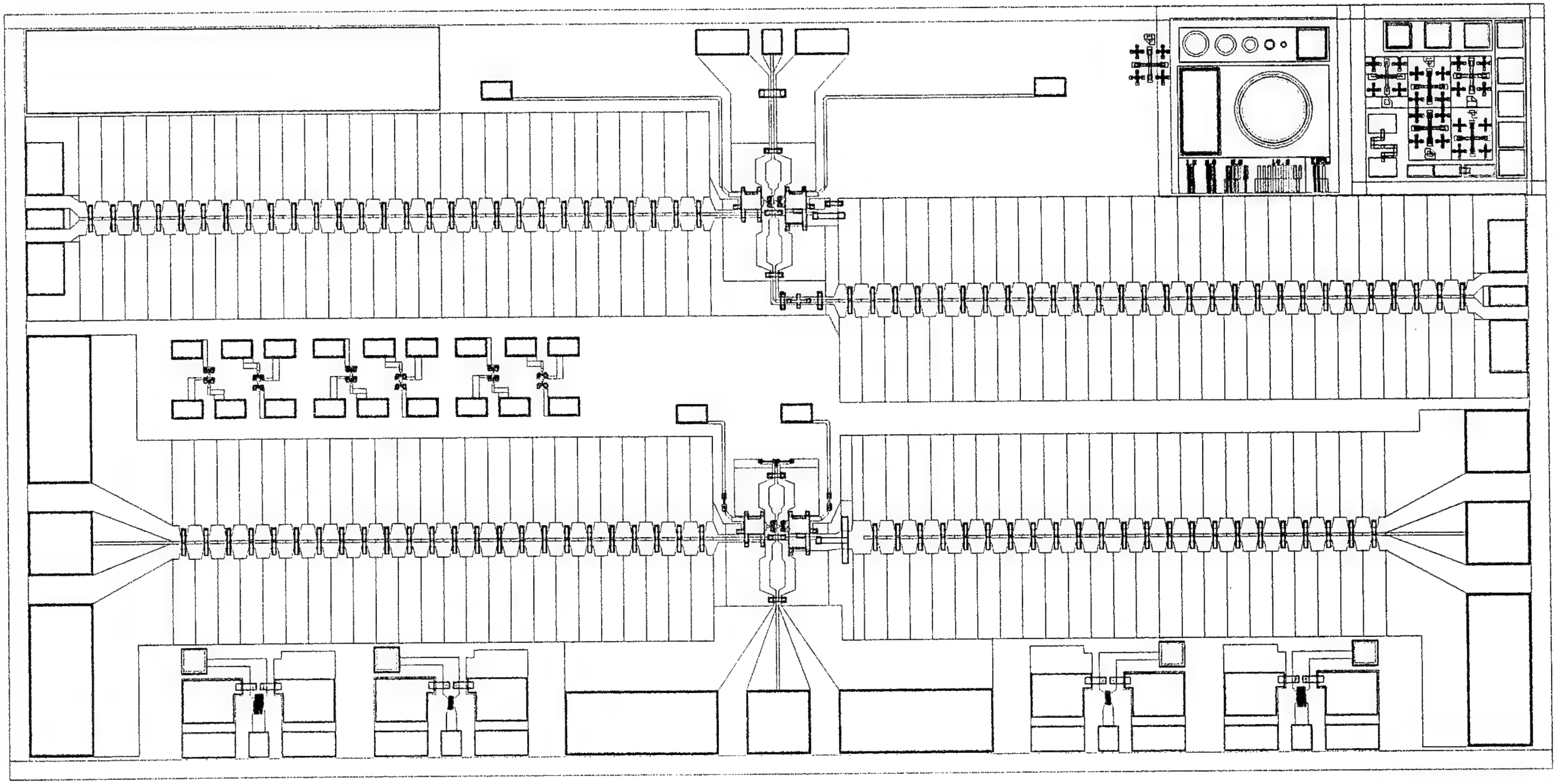
Page Size: A

Revision: fill_in_rev fill_in_date Page 1 of 1

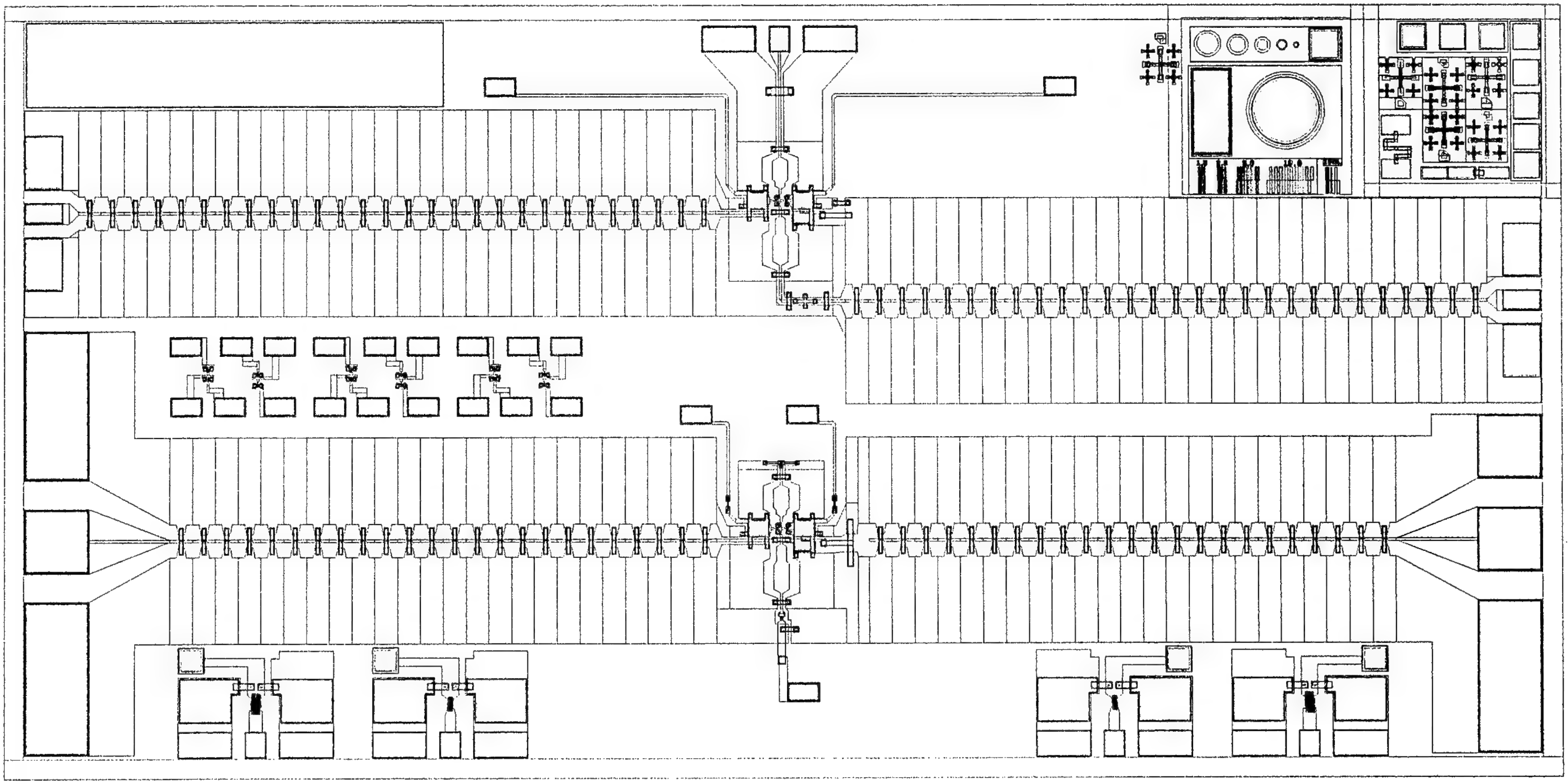
Appendix 8 – Sampler Physical Layout

NLTL Sampler (3000x6000um)
Photodiode Sampler (3000x6000um)

NLTL SAMPLER



PD SAMPLER

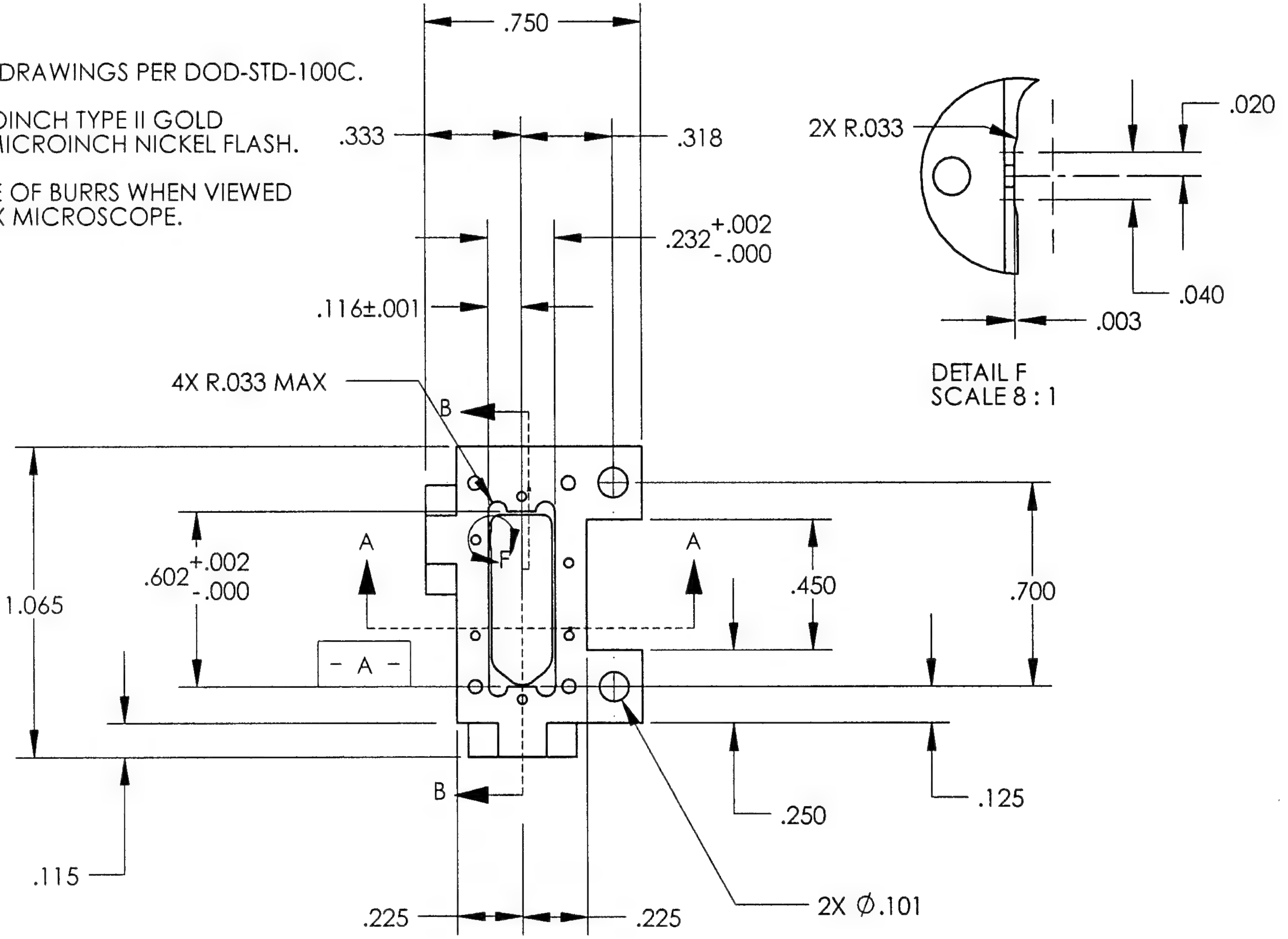


Appendix 9 – Microwave housing drawings

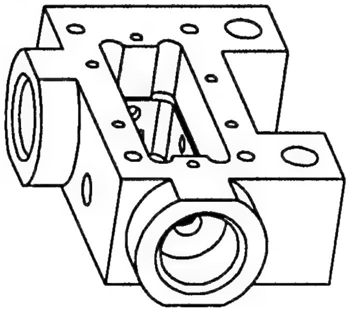
Sampler Housing (five drawings)
Sampler top cover (one drawing)
Sampler bottom cover (one drawing)

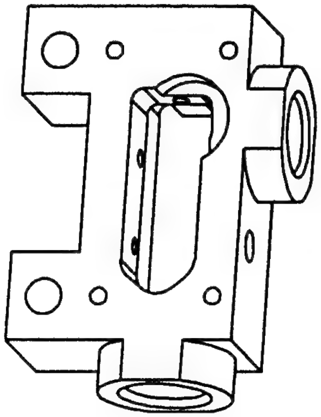
NOTES:

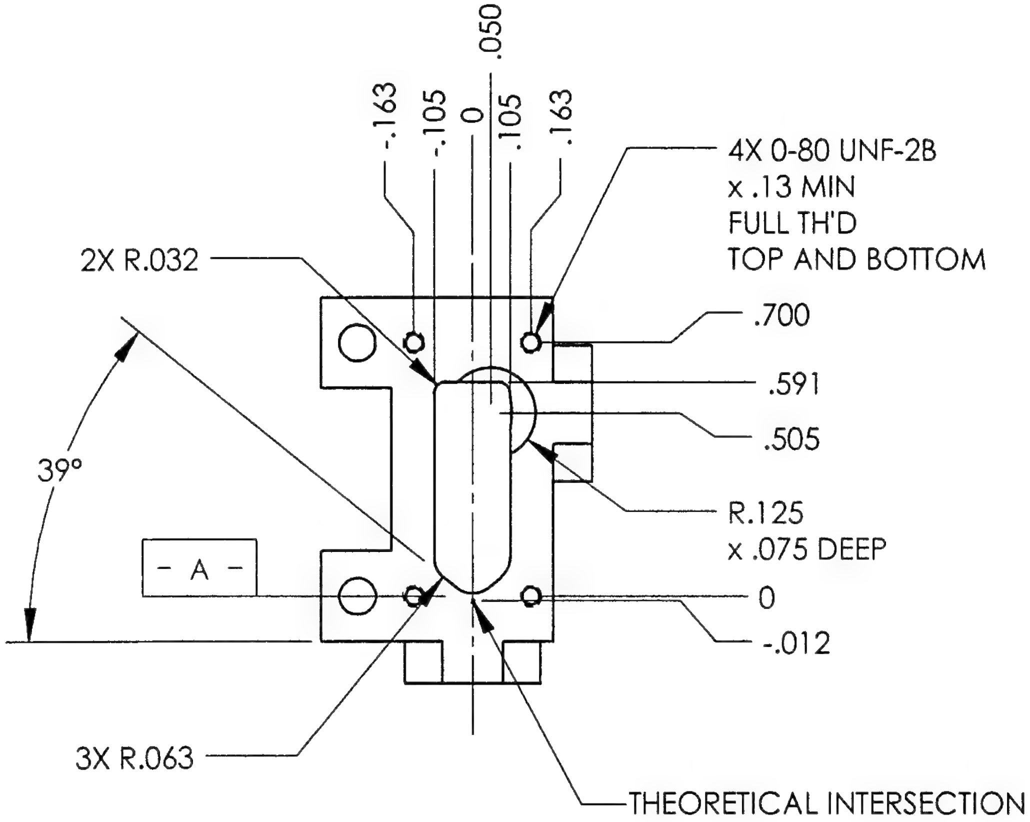
1. INTERPRET DRAWINGS PER DOD-STD-100C.
2. 100 MICROINCH TYPE II GOLD
   OVER 50 MICROINCH NICKEL FLASH.
3. TO BE FREE OF BURRS WHEN VIEWED
   UNDER 15X MICROSCOPE.

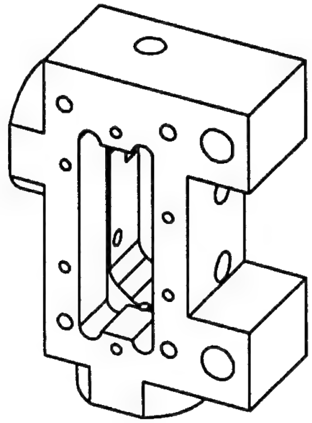


MATERIAL 360 BRASS	FINISH SEE NOTES	SCALE 2/1	SAMPLER HOUSING K & V CONNS
TOLERANCES: .XX ± .01 .XXX ± .005	UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. TOLERANCES PER ANSI Y14.5M 1982 AND NFI SPEC 01-004.	NEW FOCUS, INC SUNNYVALE, CALIFORNIA	DRAWN JPW 2/25/98 FILE NAME PROJ ENGR GRE/ACD SIZE A
ANGLES ± 1°	THIRD ANGLE PROJECTION	©1993 NEW FOCUS INC. THIS INFORMATION MAY NOT BE COPIED OR USED WITHOUT THE EXPRESS WRITTEN PERMISSION OF NEW FOCUS.	DWG NO. FR001044 REV. A SHEET 1 OF 5

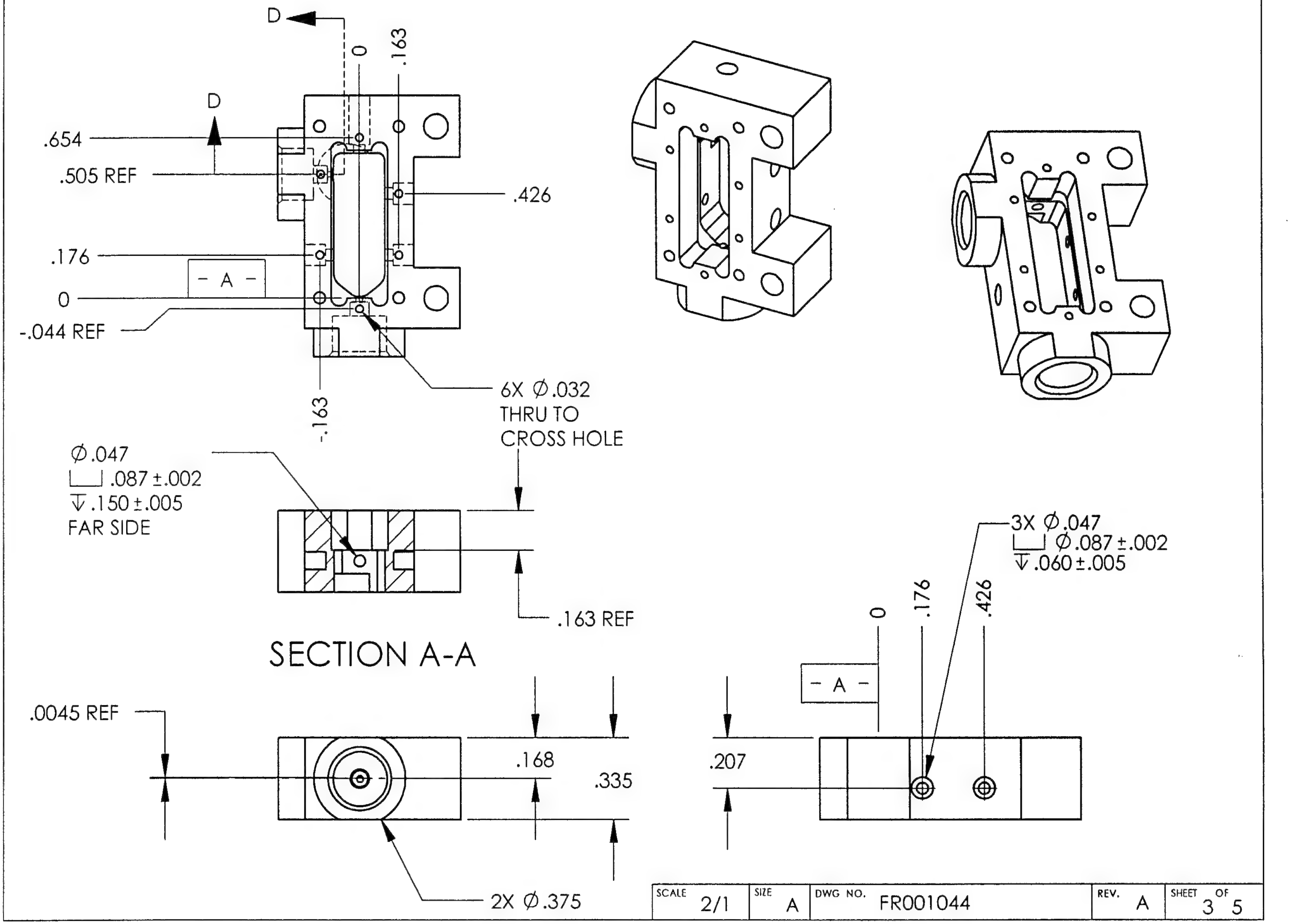




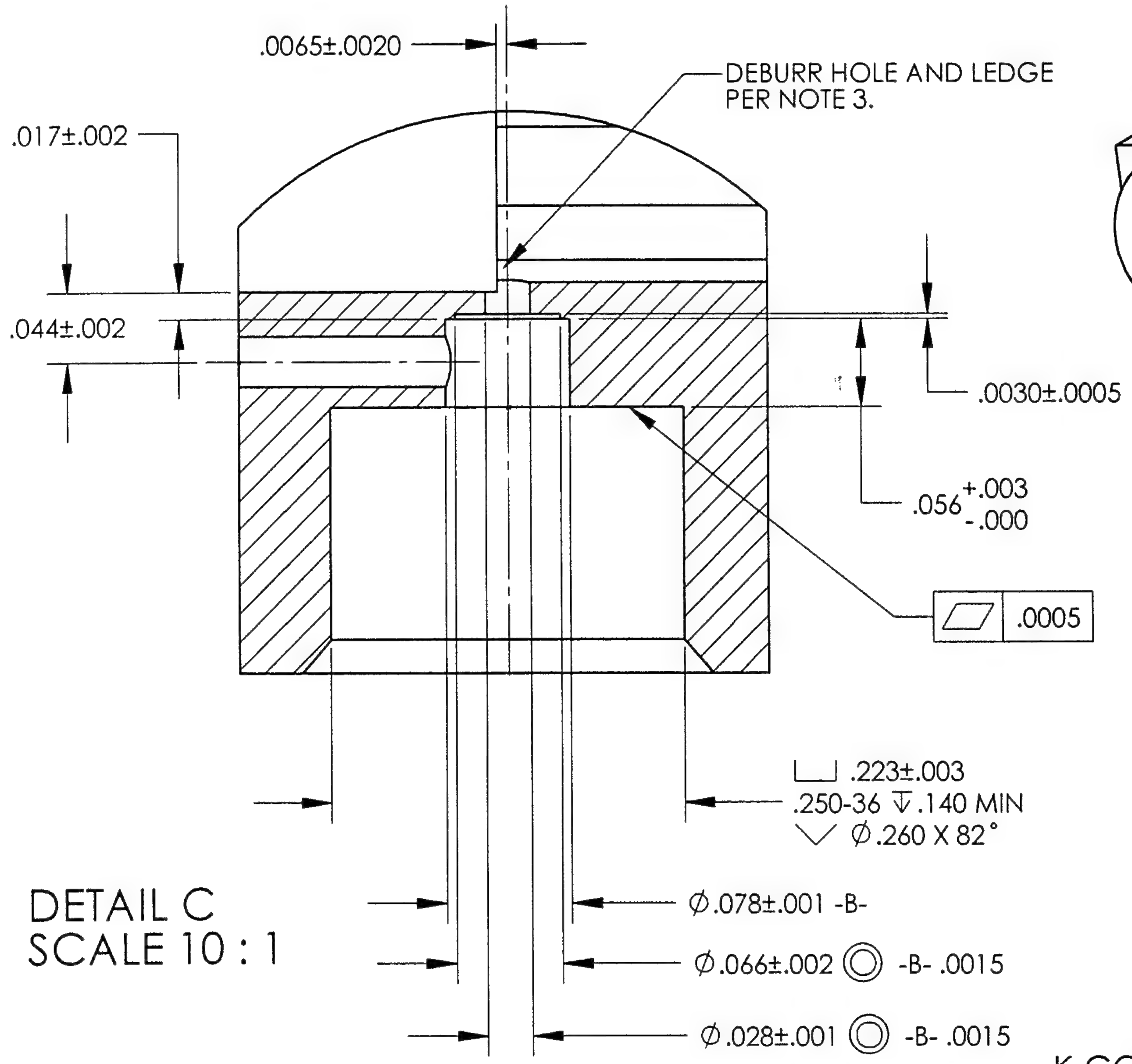




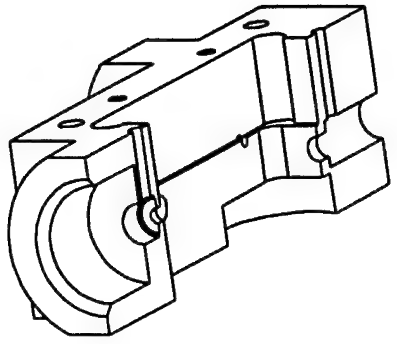
SCALE	SIZE	DWG NO.	REV.	SHEET OF
2/1	A	FR001044	A	2 5

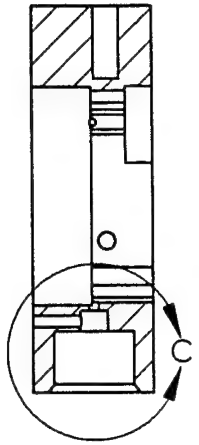


SCALE	SIZE	DWG NO.	REV.	SHEET OF
2/1	A	FR001044	A	3 5



DETAIL C
SCALE 10 : 1

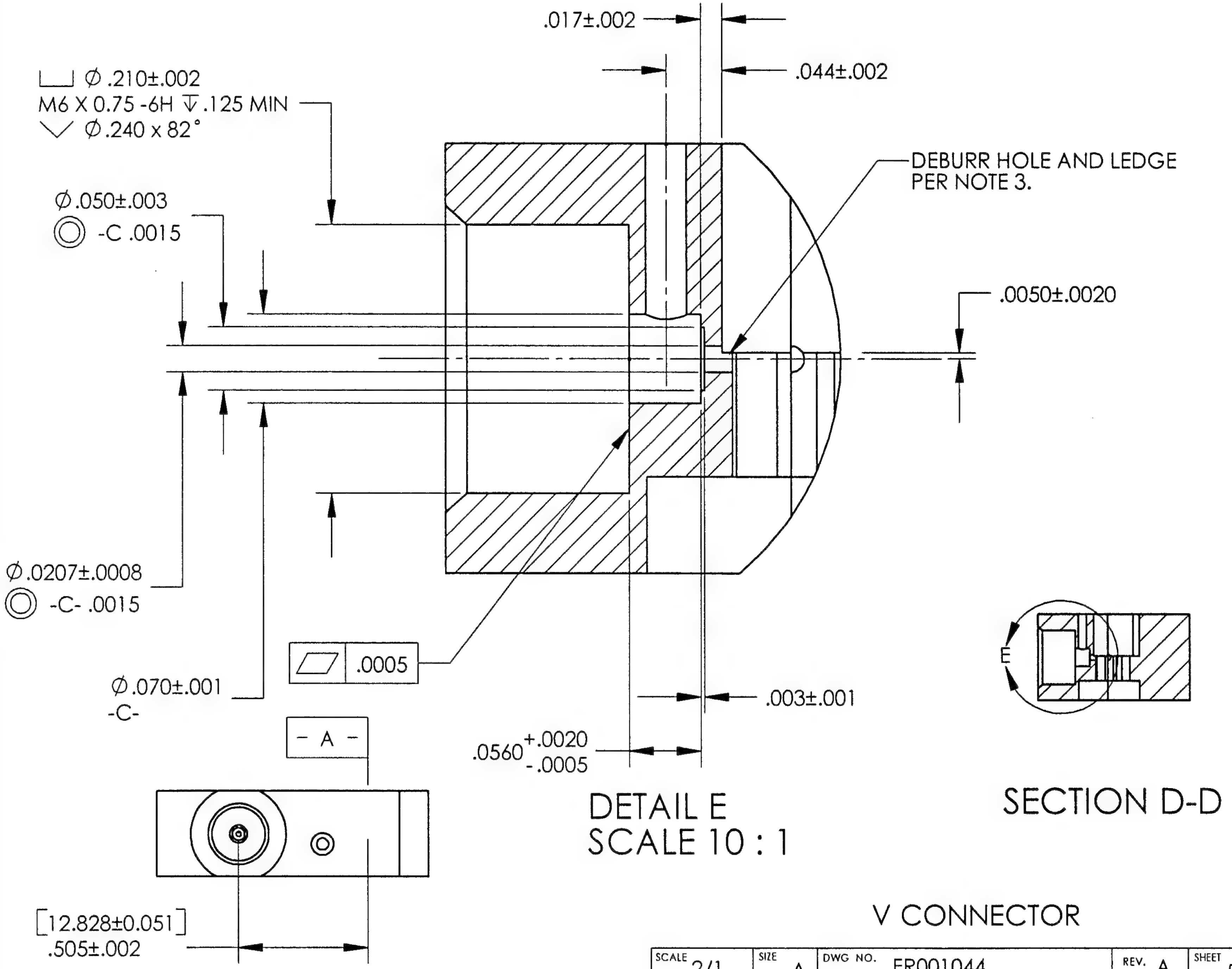




SECTION B-B

K CONNECTOR

SCALE	SIZE	DWG NO.	REV.	SHEET	OF
2/1	A	FR001044	A	4	5



DETAIL E
SCALE 10 : 1

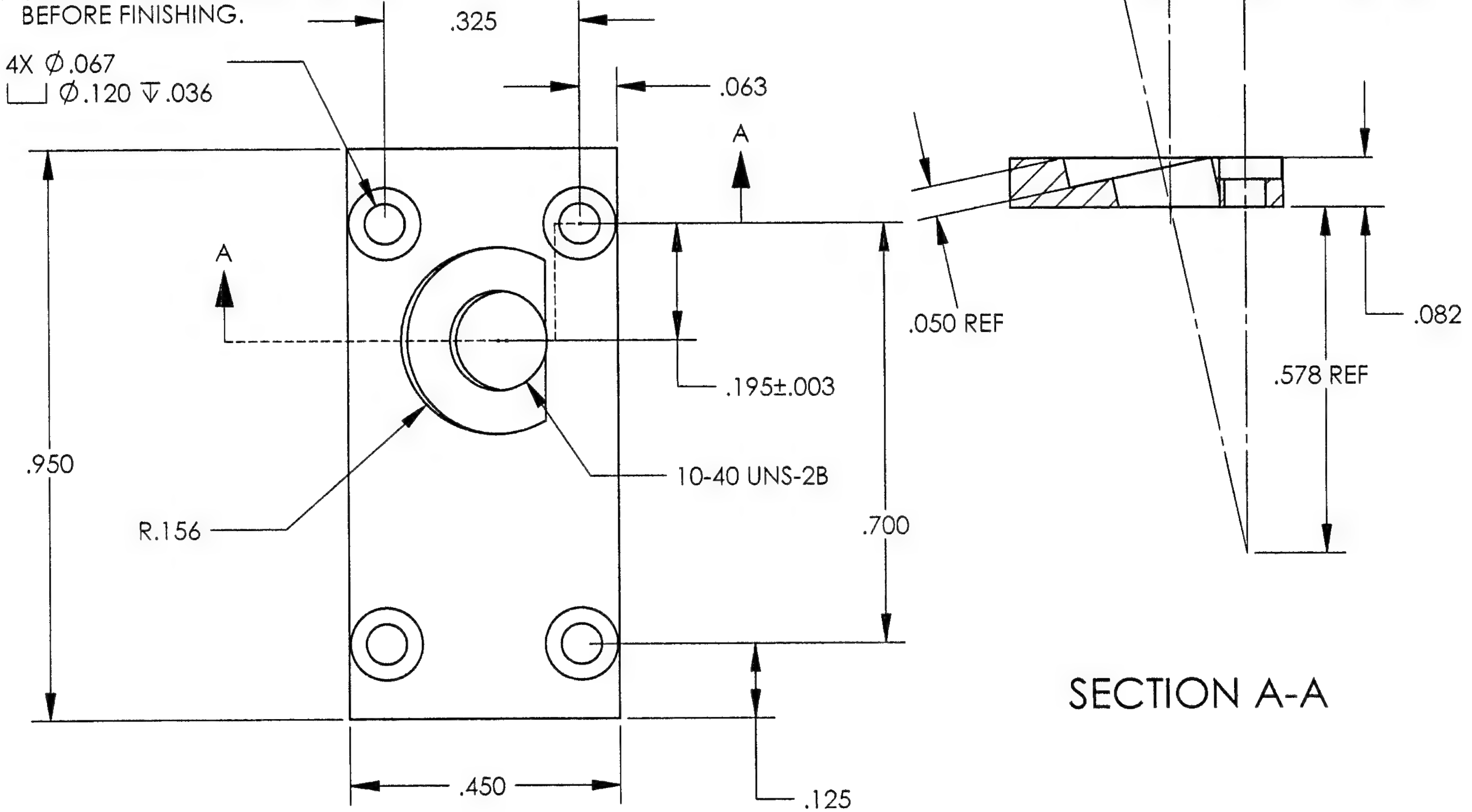
SECTION D-D

V CONNECTOR

SCALE	SIZE	DWG NO.	REV.	SHEET OF
2/1	A	FR001044	A	5 OF 5

NOTES:

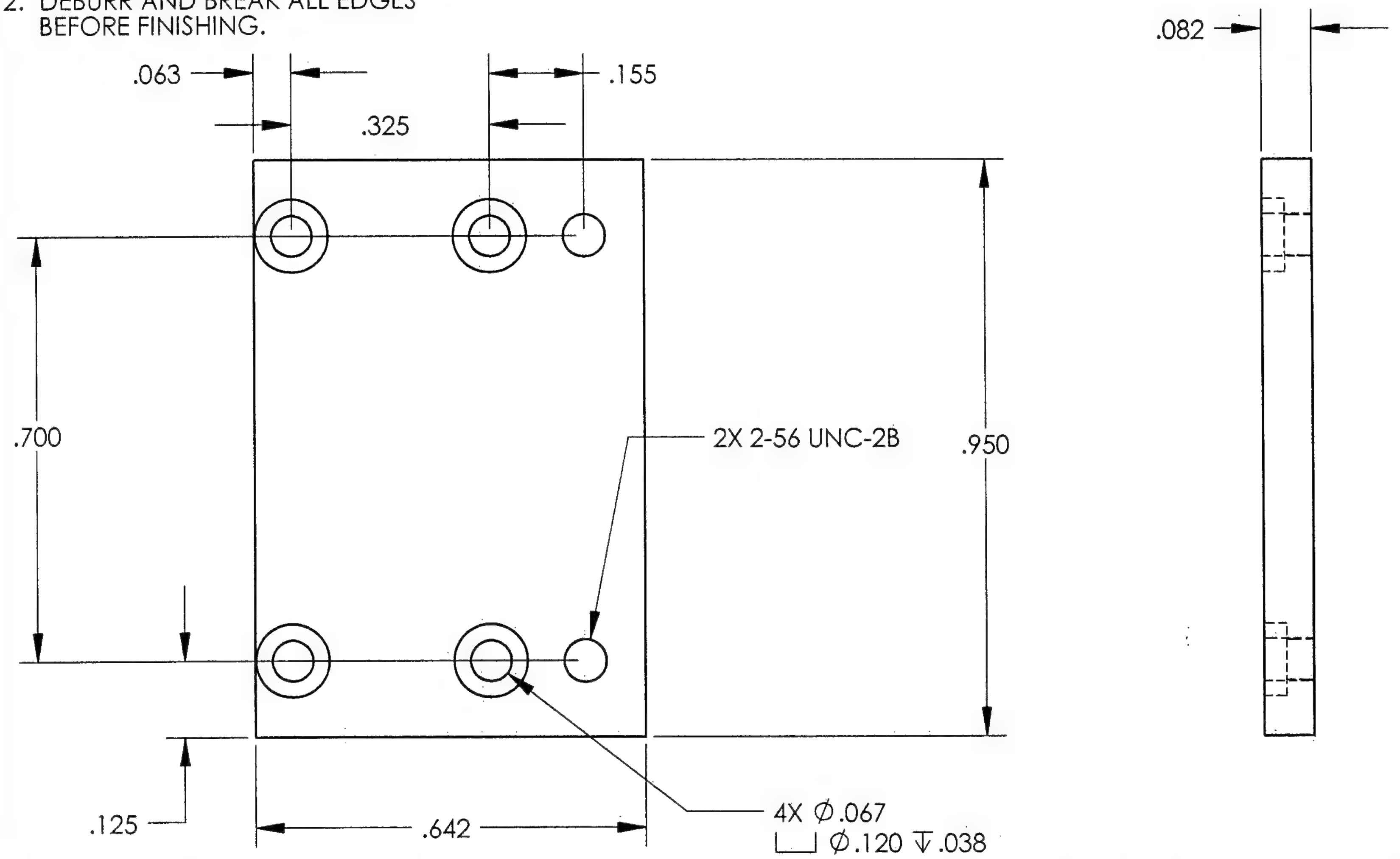
1. 100 MICROINCH TYPE II GOLD OVER 50 MICROINCH NICKEL FLASH.
2. DEBURR AND BREAK ALL EDGES BEFORE FINISHING.



MATERIAL 360 BRASS	FINISH SEE NOTES	SCALE 4/1	COVER MICROWAVE HOUSING
TOLERANCES: .XX ± .01 .XXX ± .005 ANGLES ± 1°	[V] NEW FOCUS, INC SUNNYVALE, CALIFORNIA		DRAWN JPW 2/25/98 PROJ ENGR GRE
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. TOLERANCES PER ANSI Y14.5M 1982 AND NFI SPEC 01-004.			FILE NAME SIZE A
THIRD ANGLE PROJECTION	©1993 NEW FOCUS INC. THIS INFORMATION MAY NOT BE COPIED OR USED WITHOUT THE EXPRESS WRITTEN PERMISSION OF NEW FOCUS.		DWG NO. ~~5432~~ FR001045 · REV. B · SHEET 1 OF 1

NOTES:

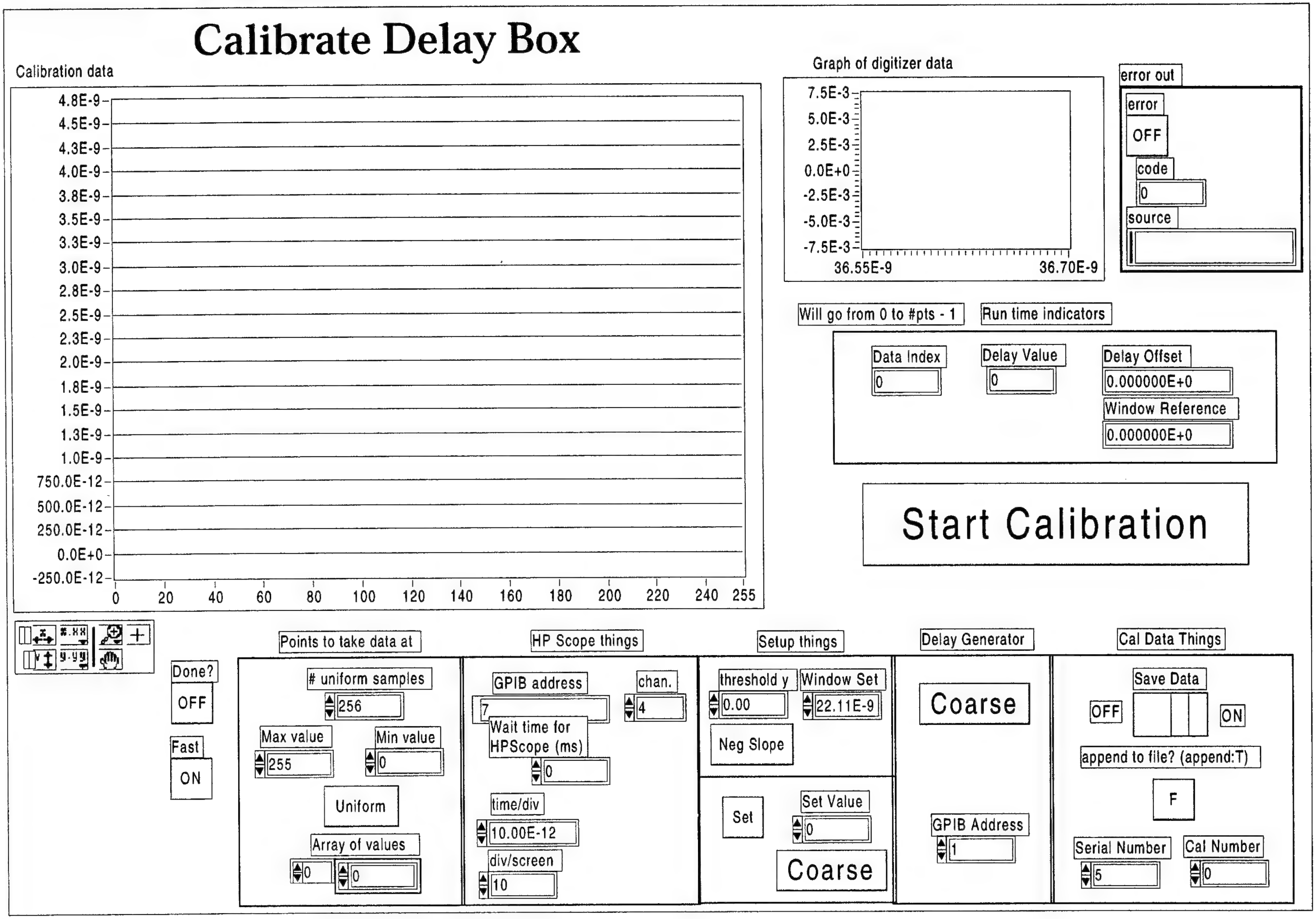
1. 100 MICROINCH TYPE II GOLD OVER 50 MICROINCH NICKEL FLASH.
2. DEBURR AND BREAK ALL EDGES BEFORE FINISHING.



MATERIAL 360 BRASS	FINISH SEE NOTES	SCALE 4/1	BOTTOM COVER µWAVE HOUSING
TOLERANCES: .XX ± .01 .XXX ± .005 ANGLES ± 1°	UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. TOLERANCES PER ANSI Y14.5M 1982 AND NFI SPEC 01-004.	NEW FOCUS, INC SUNNYVALE, CALIFORNIA	DRAWN JPW 2/25/98
			PROJ ENGR GRE
	THIRD ANGLE PROJECTION	©1993 NEW FOCUS INC. THIS INFORMATION MAY NOT BE COPIED OR USED WITHOUT THE EXPRESS WRITTEN PERMISSION OF NEW FOCUS.	DWG NO. ~~7654~~ FR001046

FILE NAME	
SIZE	A
REV.	B
SHEET	1 OF 1

Appendix 10 – Labview Calibration



Calibrate Delay Box

error out
error: OFF
code: 0
source:

Will go from 0 to #pts - 1

Run time indicators
Data Index: 0
Delay Value: 0
Delay Offset: 0.000000E+0
Window Reference: 0.000000E+0

Start Calibration

Done?: OFF
Fast: ON

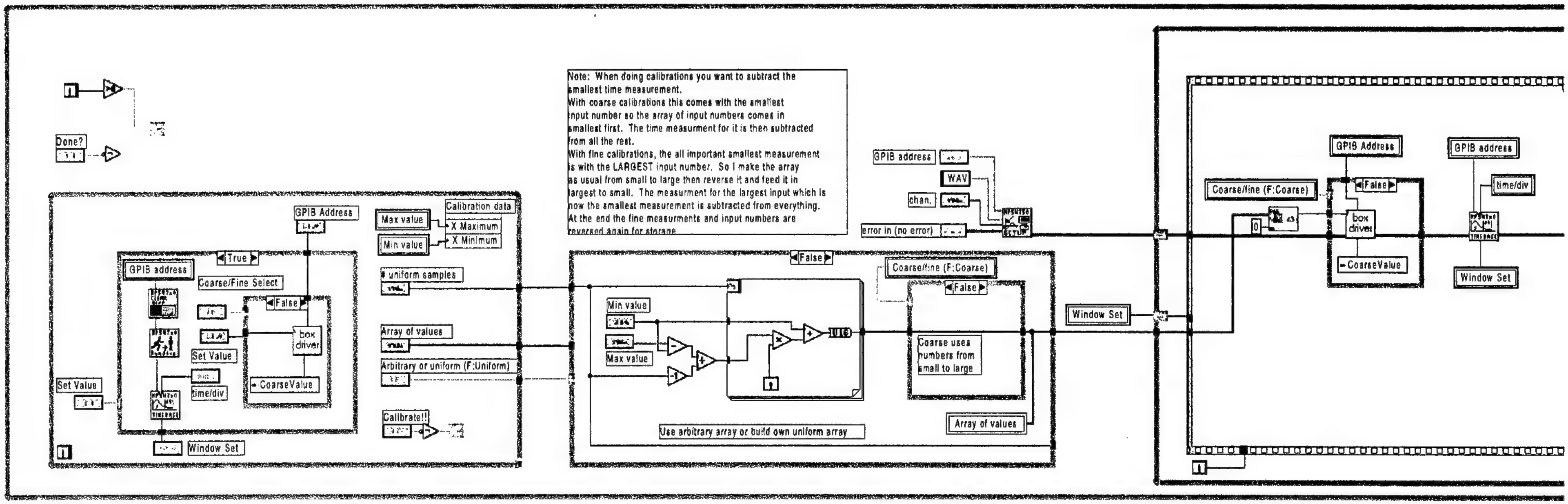
Points to take data at
# uniform samples: 256
Max value: 255
Min value: 0
Uniform
Array of values: 0 | 0

HP Scope things
GPIB address: 7
chan.: 4
Wait time for HPScope (ms): 0
time/div: 10.00E-12
div/screen: 10

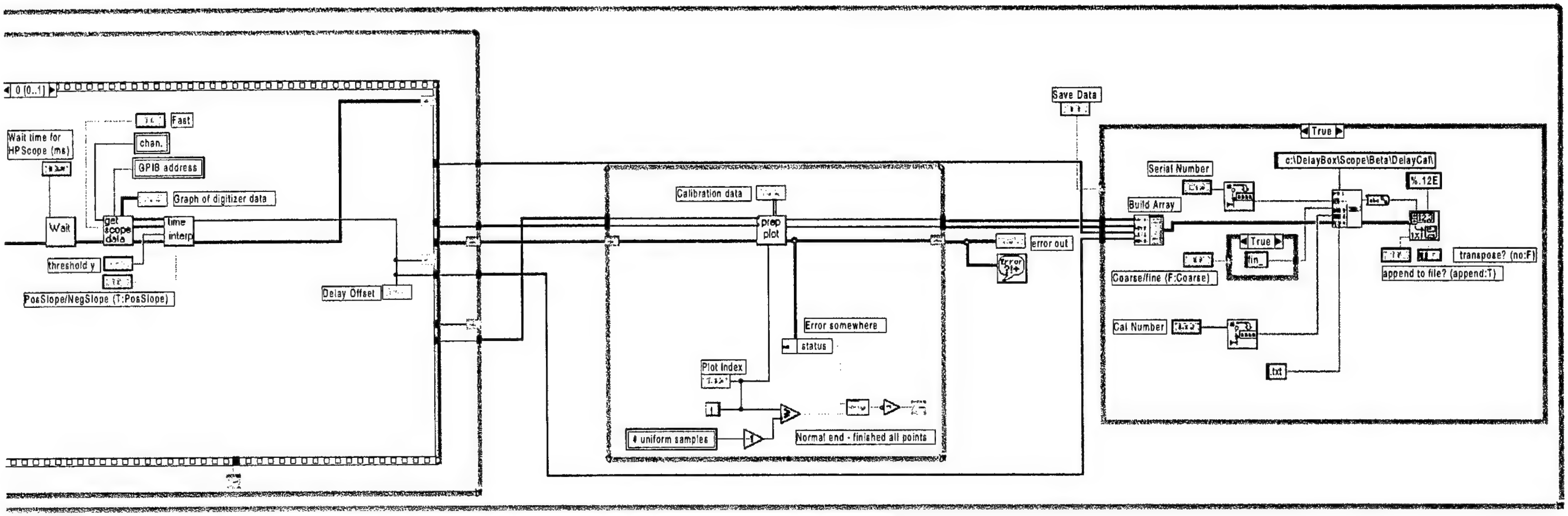
Setup things
threshold y: 0.00
Window Set: 22.11E-9
Neg Slope
Set
Set Value: 0
Coarse

Delay Generator
Coarse
GPIB Address: 1

Cal Data Things
Save Data: OFF ON
append to file? (append:T): F
Serial Number: 5
Cal Number: 0

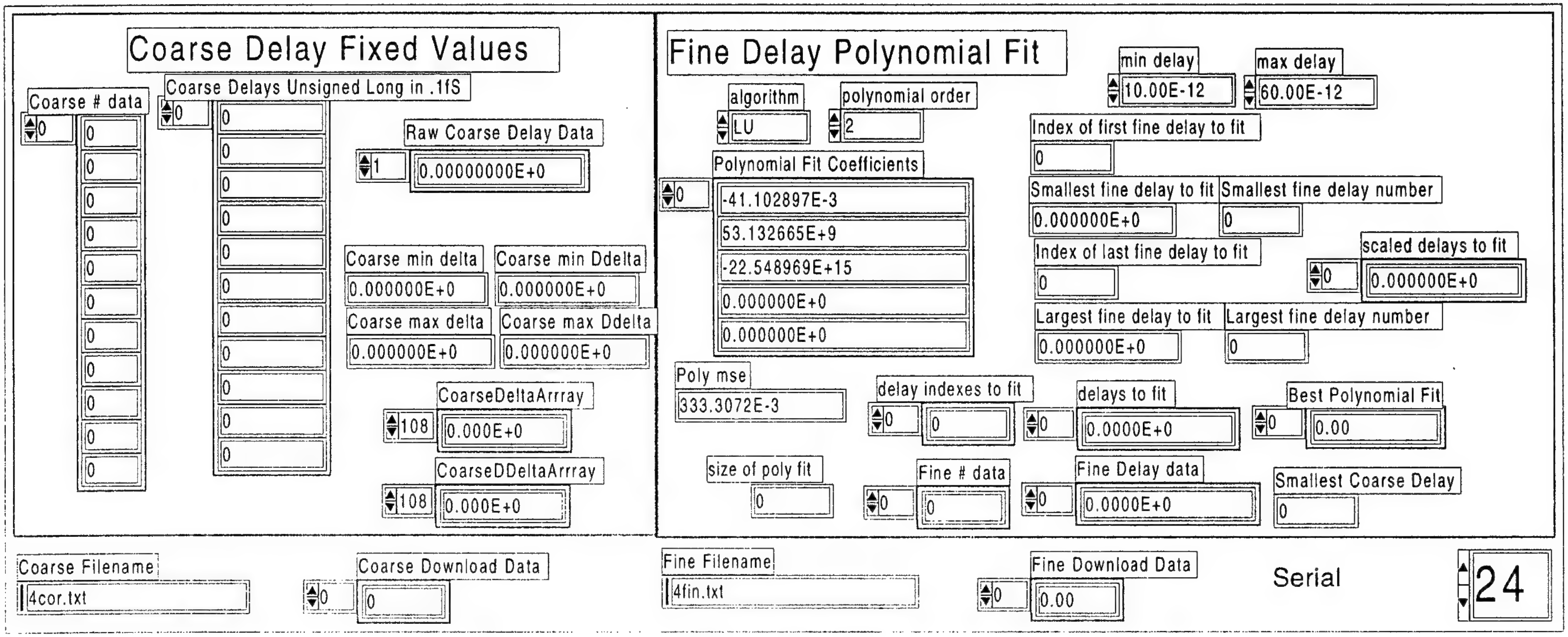


Note: When doing calibrations you want to subtract the smallest time measurement.
With coarse calibrations this comes with the smallest input number so the array of input numbers comes in smallest first. The time measurment for it is then subtracted from all the rest.
With fine calibrations, the all important smallest measurement is with the LARGEST input number. So I make the array as usual from small to large then reverse it and feed it in largest to small. The measurment for the largest input which is now the smallest measurement is subtracted from everything.
At the end the fine measurments and input numbers are reversed again for storage



0 [0..1]
Wait time for HPScope (ms)
Wait
Fast
chan.
GPIB address
Graph of digitizer data
get scope data
time interp
threshold y
PosSlope/NegSlope (T:PosSlope)
Delay Offset
Calibration data
prep plot
Error somewhere
status
Plot Index
# uniform samples
Normal end - finished all points
error out
Save Data
True
Serial Number
Build Array
c:\DelayBox\Scope\Beta\DelayCal\
%.12E
True
fin_
Coarse/fine (F:Coarse)
transpose? (no:F)
append to file? (append:T)
Cal Number
.txt

Appendix 11 – Labview Coefficient Generation



Coarse Delay Fixed Values

Coarse # data: ◆0
0
0
0
0
0
0
0
0
0
0
0

Coarse Delays Unsigned Long in .1fS: ◆0
0
0
0
0
0
0
0
0
0
0
0

Raw Coarse Delay Data: ◆1 0.00000000E+0

Coarse min delta: 0.000000E+0

Coarse min Ddelta: 0.000000E+0

Coarse max delta: 0.000000E+0

Coarse max Ddelta: 0.000000E+0

CoarseDeltaArrray: ◆108 0.000E+0

CoarseDDeltaArrray: ◆108 0.000E+0

Fine Delay Polynomial Fit

algorithm: ◆LU

polynomial order: ◆2

min delay: ◆10.00E-12

max delay: ◆60.00E-12

Polynomial Fit Coefficients: ◆0
-41.102897E-3
53.132665E+9
-22.548969E+15
0.000000E+0
0.000000E+0

Index of first fine delay to fit: 0

Smallest fine delay to fit: 0.000000E+0

Smallest fine delay number: 0

Index of last fine delay to fit: 0

scaled delays to fit: ◆0 0.000000E+0

Largest fine delay to fit: 0.000000E+0

Largest fine delay number: 0

Poly mse: 333.3072E-3

delay indexes to fit: ◆0 0

delays to fit: ◆0 0.0000E+0

Best Polynomial Fit: ◆0 0.00

size of poly fit: 0

Fine # data: ◆0 0

Fine Delay data: ◆0 0.0000E+0

Smallest Coarse Delay: 0

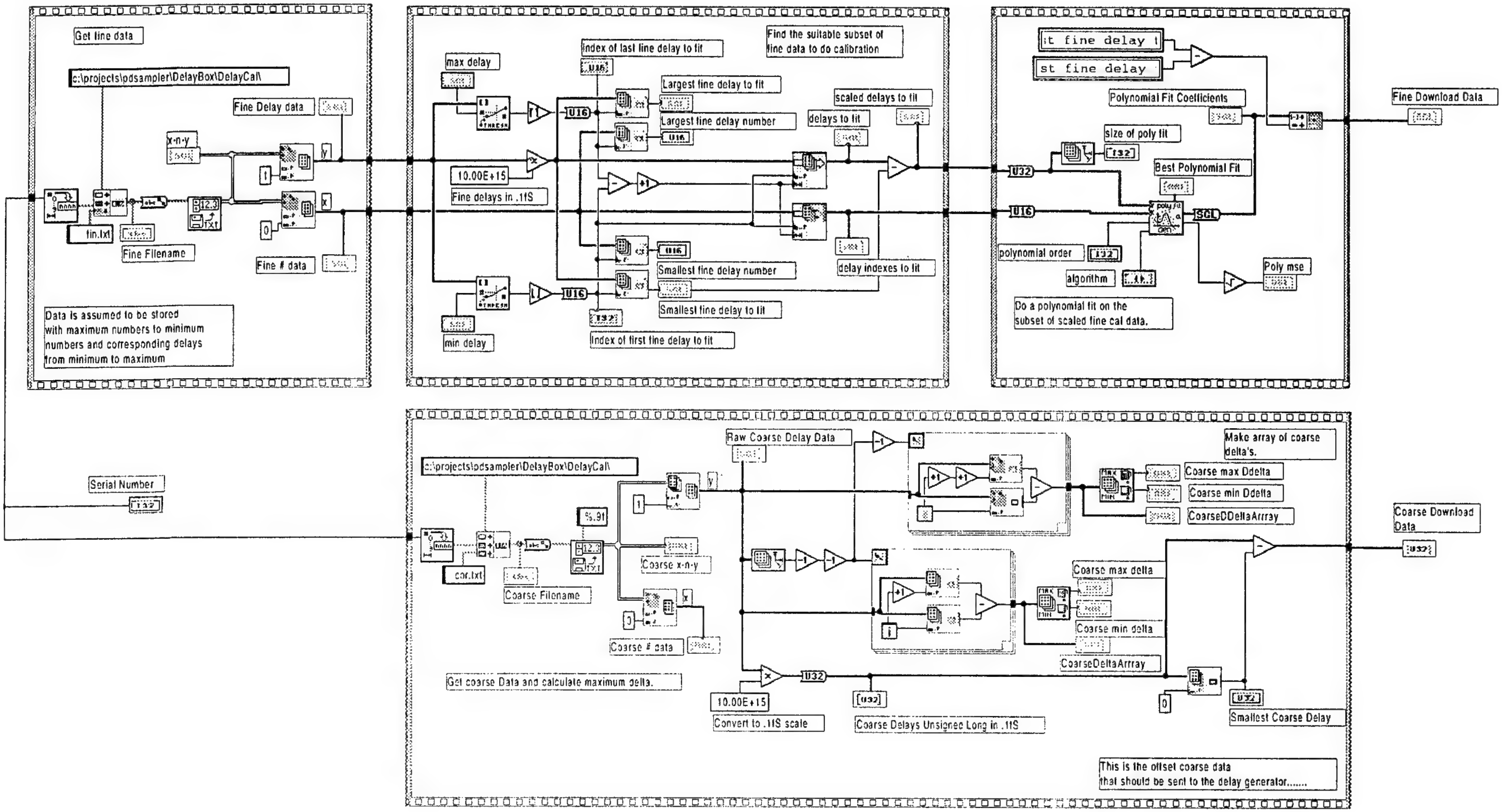
Coarse Filename: 4cor.txt

Coarse Download Data: ◆0 0

Fine Filename: 4fin.txt

Fine Download Data: ◆0 0.00

Serial ◆24



Get fine data

Data is assumed to be stored
with maximum numbers to minimum
numbers and corresponding delays
from minimum to maximum

Find the suitable subset of
fine data to do calibration

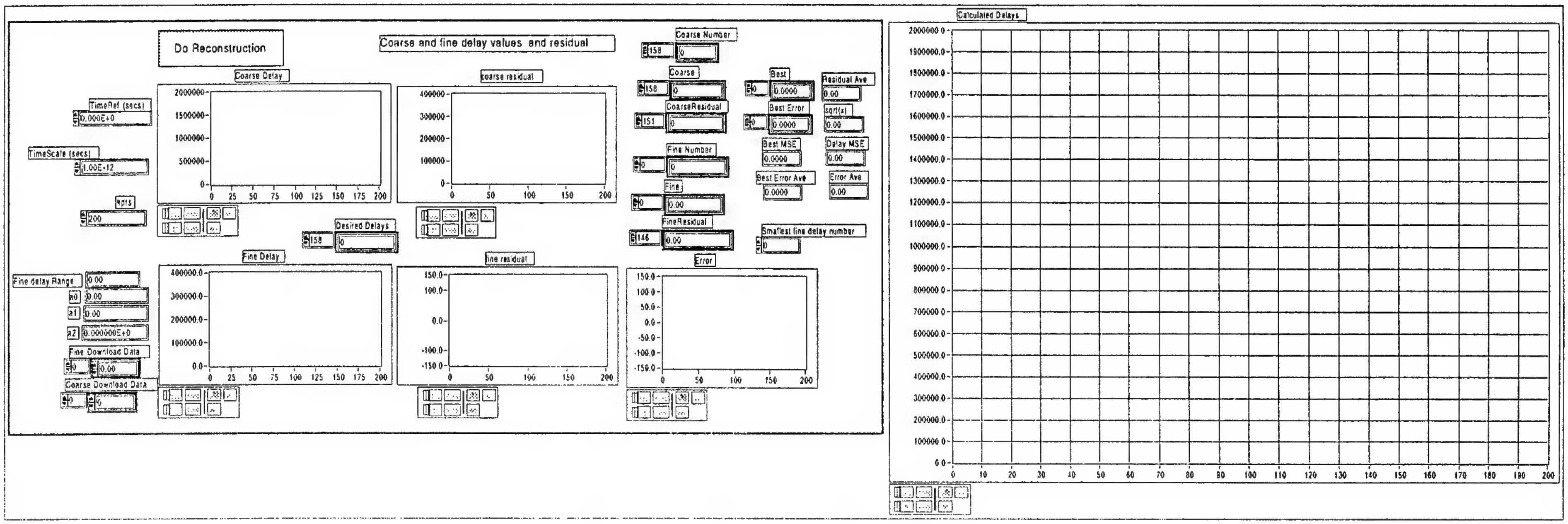
Do a polynomial fit on the
subset of scaled fine cal data.

Get coarse Data and calculate maximum delta.

Make array of coarse
delta's.

This is the offset coarse data
that should be sent to the delay generator.......

Appendix 12 – Labview Data Verification



TimeRef (secs) 0.000E+0

TimeScale (secs) 1.00E-12

#pts 200

Desired Delays 158 0

Fine delay Range 0.00

a0 0.00

a1 0.00

a2 0.000000E+0

Fine Download Data 0 0.00

Coarse Download Data 0 0

Coarse Number 158 0

Coarse 158 0

CoarseResidual 151 0

Fine Number 0 0

Fine 0 0.00

FineResidual 146 0.00

Best 0 0.0000

Best Error 0 0.0000

Best MSE 0.0000

Best Error Ave 0.0000

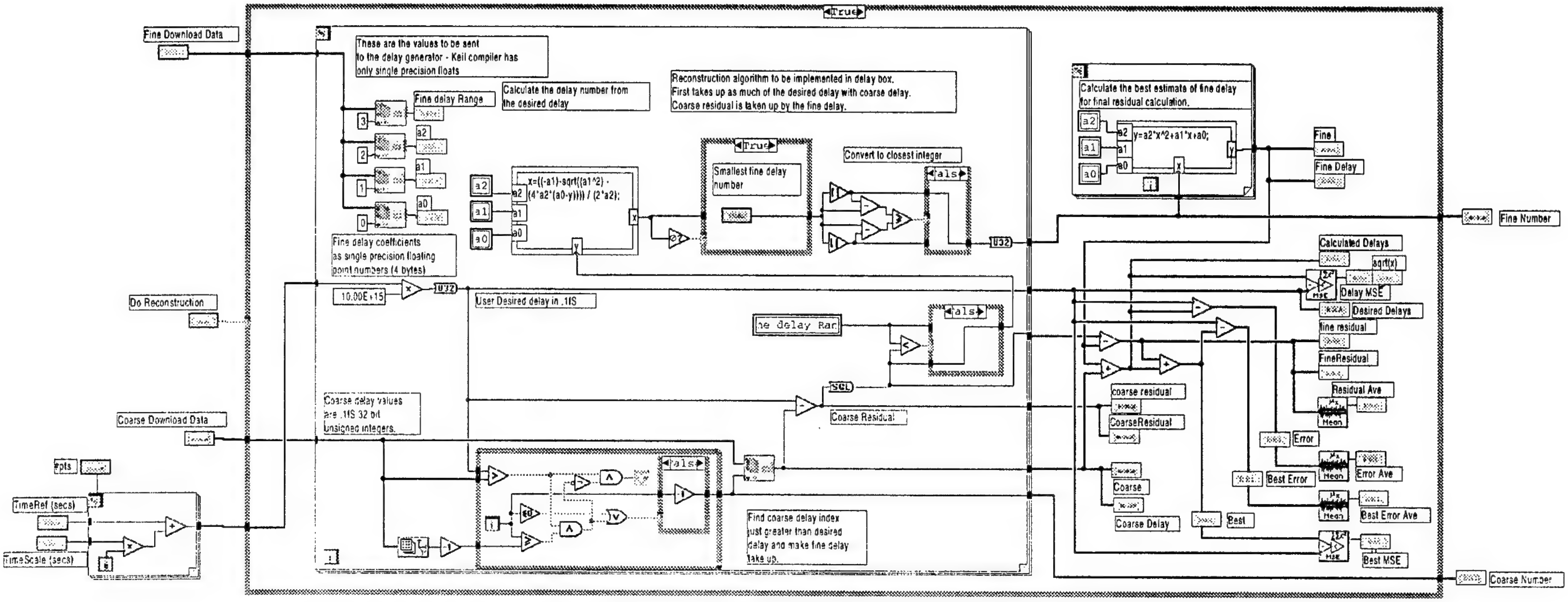
Residual Ave 0.00

sqrt(x) 0.00

Delay MSE 0.00

Error Ave 0.00

Smallest fine delay number 0



These are the values to be sent to the delay generator - Keil compiler has only single precision floats

Calculate the delay number from the desired delay

Reconstruction algorithm to be implemented in delay box. First takes up as much of the desired delay with coarse delay. Coarse residual is taken up by the fine delay.

Fine delay coefficients as single precision floating point numbers (4 bytes)

```
x=((-a1)-sqrt((a1^2) -
(4*a2*(a0-y)))) / (2*a2);
```

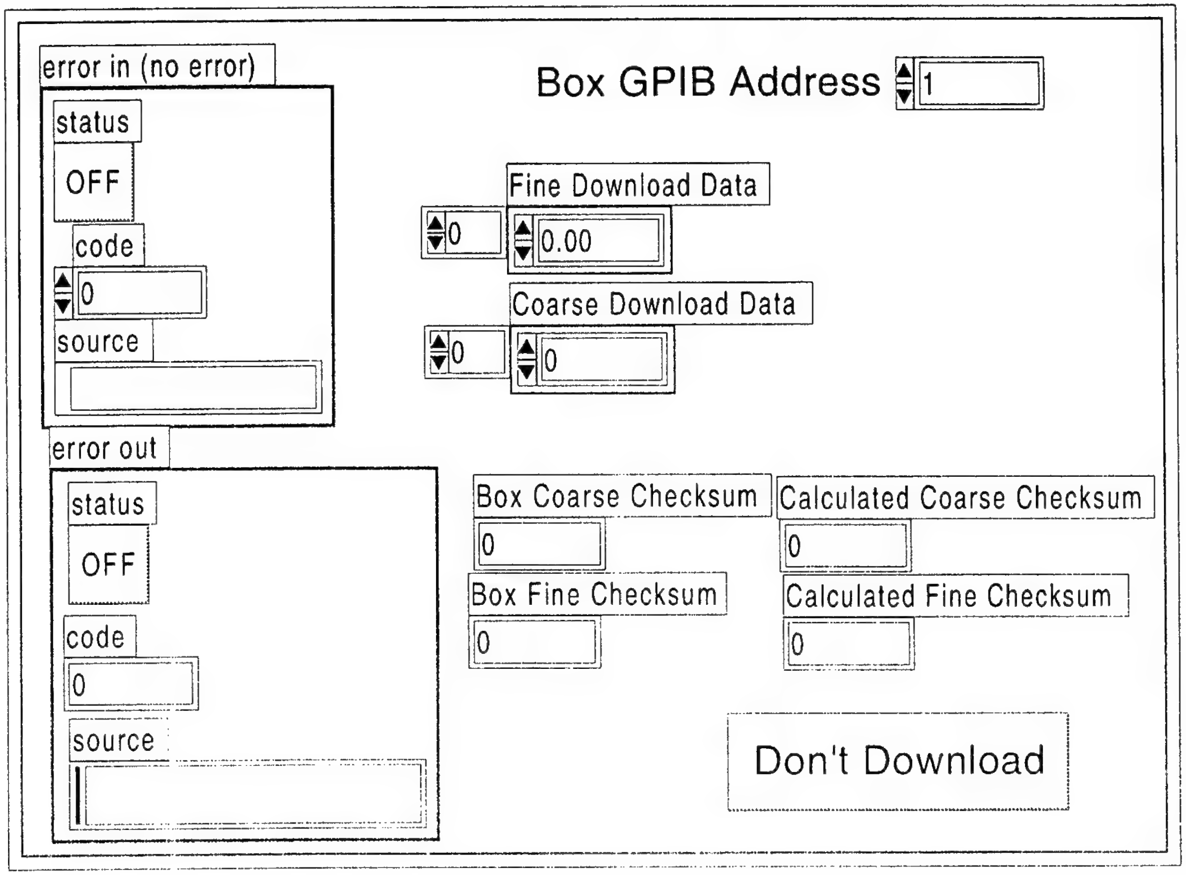
Coarse delay values are .1fS 32 bit unsigned integers.

Find coarse delay index just greater than desired delay and make fine delay take up.

Calculate the best estimate of fine delay for final residual calculation.

```
y=a2*x^2+a1*x+a0;
```

Appendix 13 – Labview Data Download



error in (no error)

status: OFF

code: 0

source:

Box GPIB Address: 1

Fine Download Data: 0 | 0.00

Coarse Download Data: 0 | 0

error out

status: OFF

code: 0

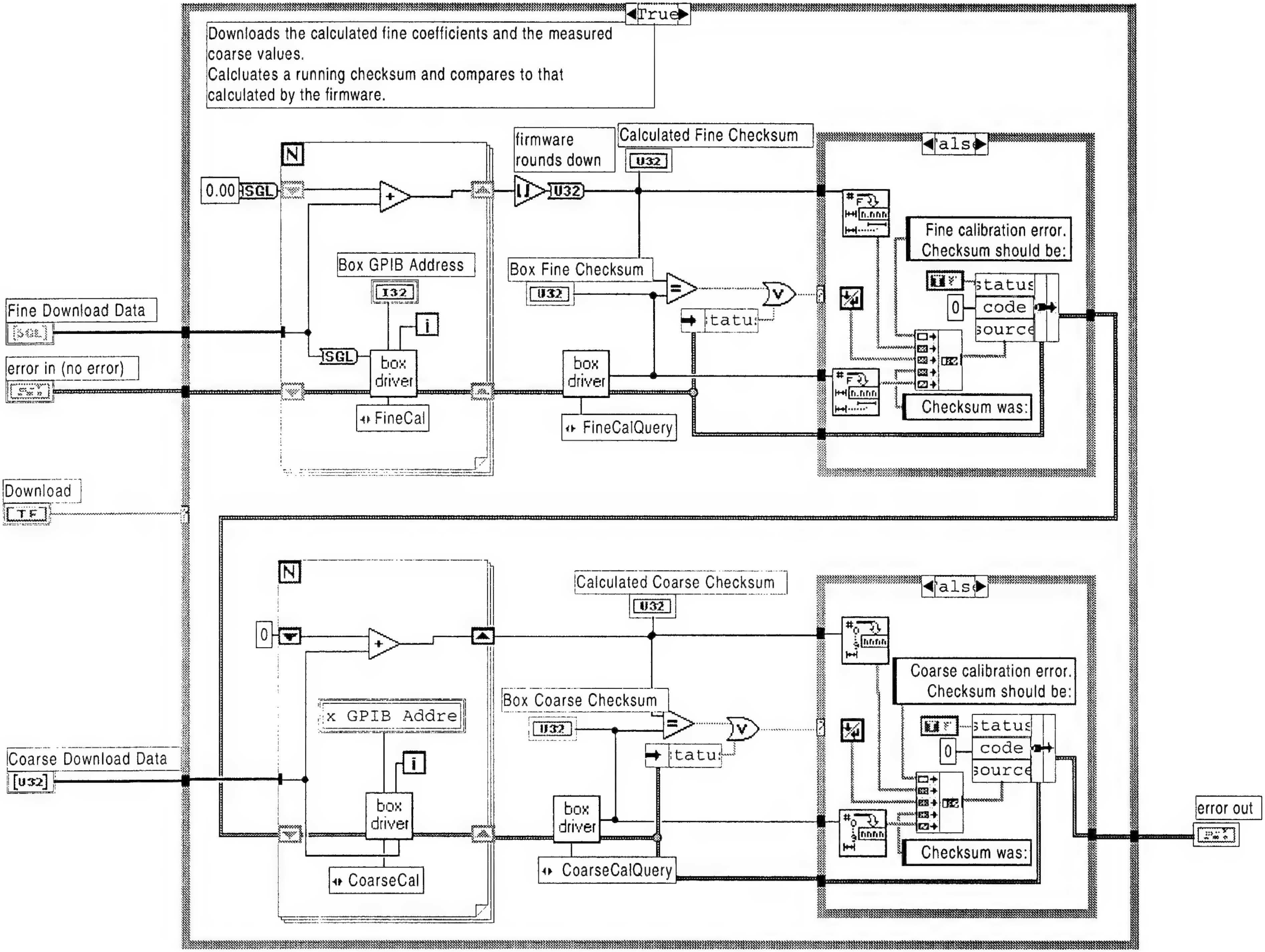
source:

Box Coarse Checksum: 0

Calculated Coarse Checksum: 0

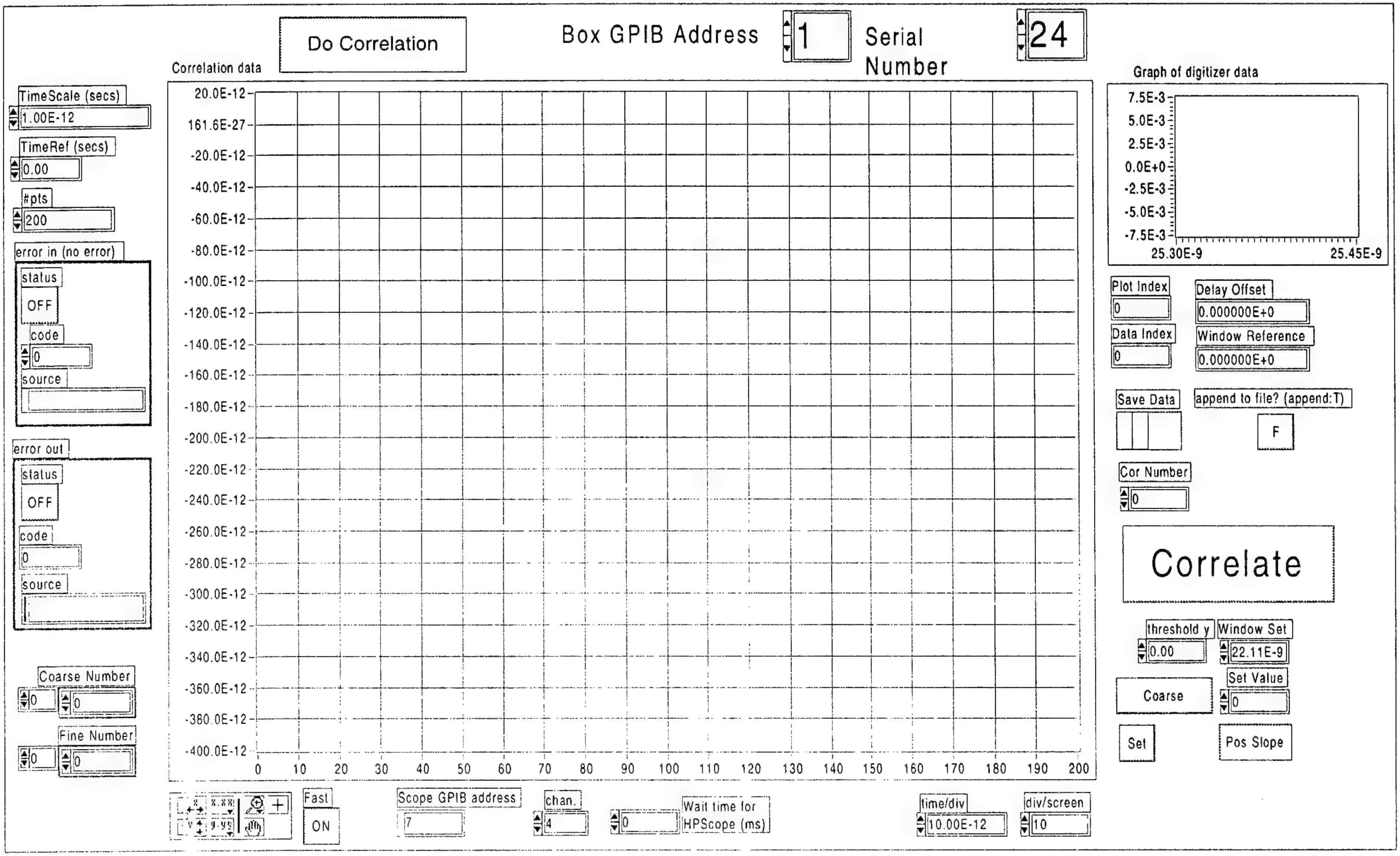
Box Fine Checksum: 0

Calculated Fine Checksum: 0



Downloads the calculated fine coefficients and the measured coarse values.
Calcluates a running checksum and compares to that calculated by the firmware.

Appendix 14 – Labview Correlation



Box GPIB Address 1

Serial Number 24

TimeScale (secs) 1.00E-12

TimeRef (secs) 0.00

#pts 200

error in (no error)
status OFF
code 0
source

error out
status OFF
code 0
source

Coarse Number 0 0

Fine Number 0 0

Fast ON

Scope GPIB address 7

chan. 4

Wait time for HPScope (ms) 0

time/div 10.00E-12

div/screen 10

Plot Index 0

Data Index 0

Delay Offset 0.000000E+0

Window Reference 0.000000E+0

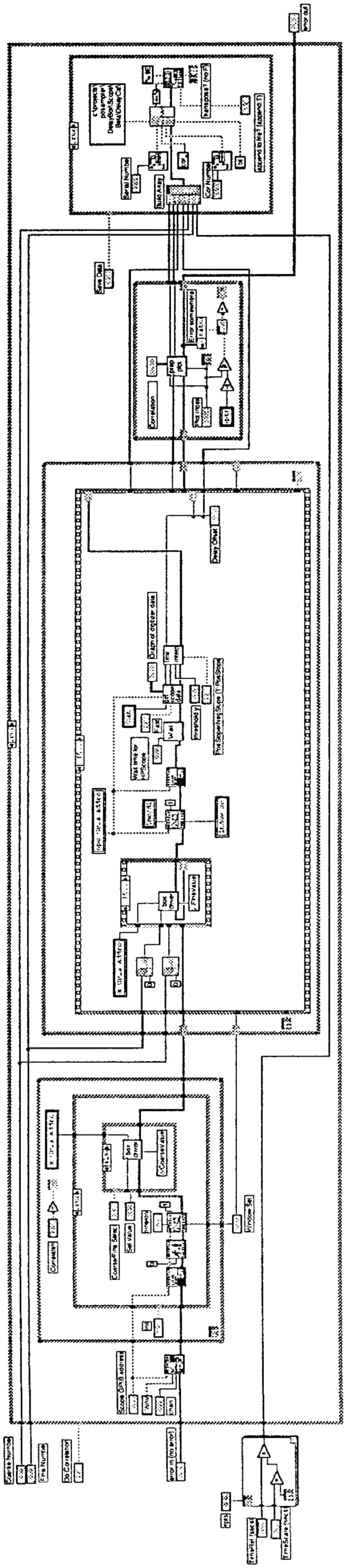
append to file? (append:T) F

Cor Number 0

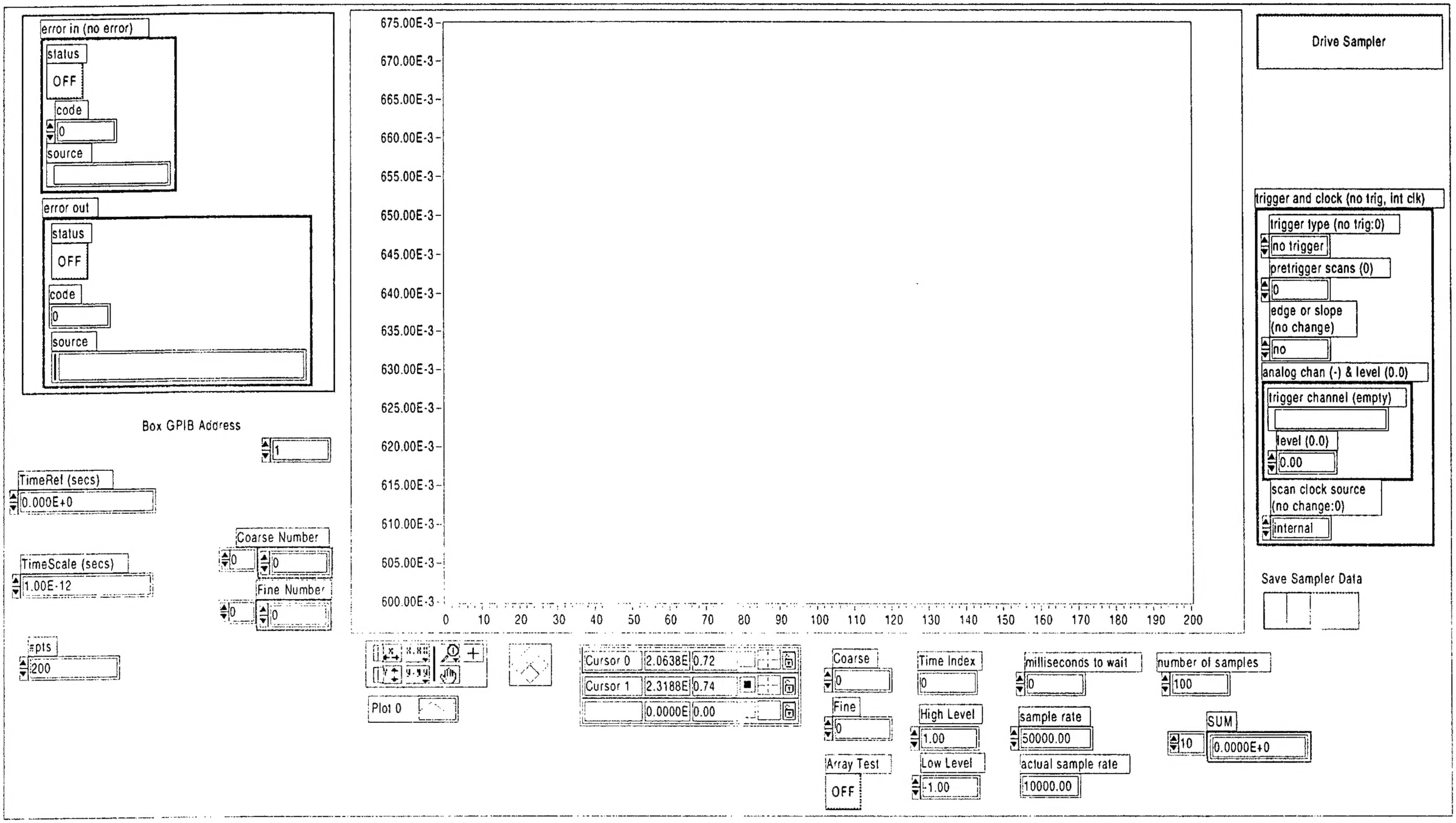
threshold y 0.00

Window Set 22.11E-9

Set Value 0


Coarse Number
Fine Number
Do Correlation
error in (no error)
Scope GPIB address
Window Set
Coarse/Fine Select
Set Value
Delay Offset
Graph of digitizer data
Wait time for HPScope
Correlation
Error somewhere
Save Data
Serial Number
Build Array
append to file? (append:T)
error out

Appendix 15 – Labview Scope



error in (no error)

status OFF

code 0

source

error out

status OFF

code 0

source

Box GPIB Address 1

TimeRel (secs) 0.000E+0

TimeScale (secs) 1.00E-12

Coarse Number 0 0

Fine Number 0 0

#pts 200

Cursor 0 2.0638E 0.72

Cursor 1 2.3188E 0.74

0.0000E 0.00

Coarse 0

Fine 0

Array Test OFF

Time Index 0

High Level 1.00

Low Level -1.00

milliseconds to wait 0

sample rate 50000.00

actual sample rate 10000.00

number of samples 100

SUM 10 0.0000E+0

Drive Sampler

trigger and clock (no trig, int clk)

trigger type (no trig:0) no trigger

pretrigger scans (0) 0

edge or slope (no change) no

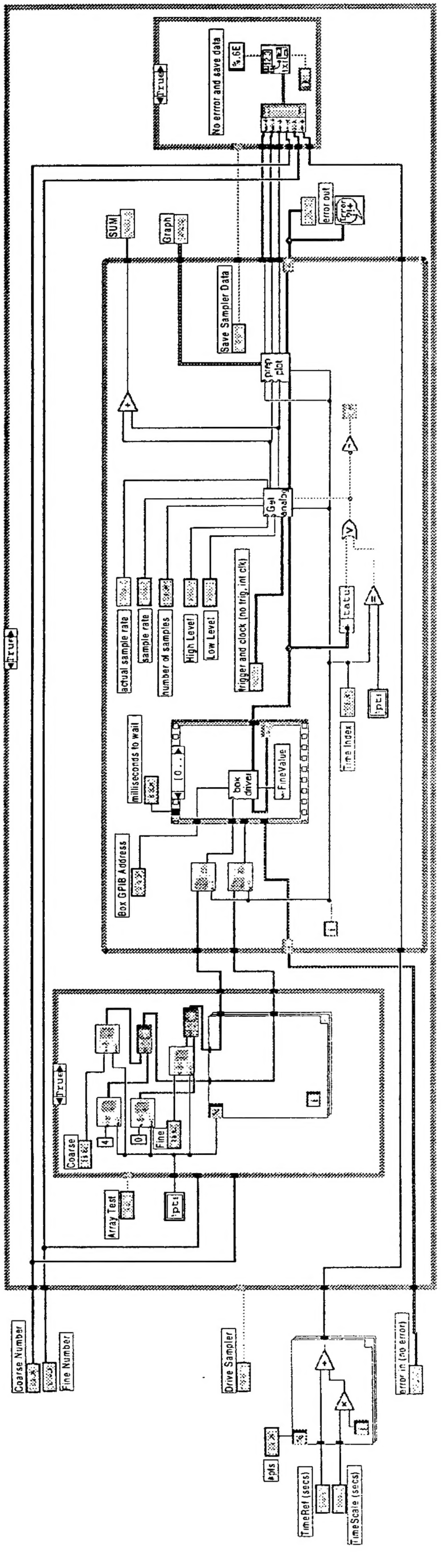
analog chan (-) & level (0.0)

trigger channel (empty)

level (0.0) 0.00

scan clock source (no change:0) internal

Save Sampler Data



Reproduced From
Best Available Copy